Als Manuskript gedruckt

Technische Universität Dresden
Herausgeber: Der Rektor

A Combinatorial Flow-based Formulation for Temporal Bin Packing Problems

John Martinovic, Nico Strasdat, José Valério de Carvalho, Fabio Furini

Preprint MATH-NM-02-2022

June 2022

A Combinatorial Flow-based Formulation for Temporal Bin Packing Problems

J. Martinovic[a,*], N. Strasdat[a], J. Valério de Carvalho[b], F. Furini[c]

[a] *Institute of Numerical Mathematics, Technische Universität Dresden, 01062 Dresden, Germany*
[b] *Departamento de Produção e Sistemas/Centro ALGORITMI, Universidade do Minho, 4710-057 Braga, Portugal*
[c] *Department of Computer, Control and Management Engineering "Antonio Ruberti", Sapienza University of Rome, 00185 Roma, Italy*

Abstract

We consider two neighboring generalizations of the classical bin packing problem: the temporal bin packing problem (TBPP) and the temporal bin packing problem with fire-ups (TBPP-FU). In both cases, the task is to arrange a set of given jobs, characterized by a resource consumption and an activity window, on homogeneous servers of limited capacity. To keep operational costs but also energy consumption low, TBPP is concerned with minimizing the number of servers in use, whereas TBPP-FU additionally takes into account the switch-on processes required for their operation. Either way, challenging integer optimization problems are obtained, which can differ significantly from each other despite the seemingly only marginal variation of the problems. In the literature, a branch-and-price method enriched with many preprocessing steps (for TBPP) and compact formulations (for TBPP-FU), benefiting from numerous reduction methods, have emerged as, currently, the most promising solution methods. In this paper, we introduce, in a sense, a unified solution framework for both problems (and, in fact, a wide variety of further interval scheduling applications) based on graph theory. Any scientific contributions in this direction failed so far because of the exponential size of the associated networks. The approach we present in this article does not change the theoretical exponentiality itself, but it can make it controllable by clever construction of the resulting graphs. In particular, for the first time all classical benchmark instances (and even larger ones) for the two problems can be solved – in times that significantly improve those of the previous approaches.

Keywords: Combinatorial Optimization, Temporal Bin Packing, Fire Ups, Interval Scheduling, Flow Formulation

1. Introduction

1.1. General Overview

The optimal assignment of given jobs to one or more servers with limited capacity is an important theoretical problem in discrete optimization, but also highly relevant in many applications from computer science [8], logistics [30], or communications engineering [14]. Despite some clear relationships between the underlying abstract problems, a wide variety of different specifications and associated terminologies have developed independently in recent years, in each of these scientific fields. To provide a coherent overview, in this article we would like to mainly focus on the operations research (or rather, the cutting and packing) perspective, but we will also refer to important concepts and results from the other areas mentioned above for further information.

In most of these scheduling problems, we consider a set of $n \in \mathbb{N}$ given jobs, each characterized by a profit

*Corresponding author
Email addresses: john.martinovic@tu-dresden.de (J. Martinovic), nico.strasdat@tu-dresden.de (N. Strasdat), vc@dps.uminho.pt (J. Valério de Carvalho), fabio.furini@uniroma1.it (F. Furini)

p_i, a resource consumption c_i, and an activity interval $[s_i, e_i)$ with $s_i < e_i$, that have to be assigned to a single- or multi-server architecture the capacity C of which has to be respected at all instants of time. Without loss of generality, all these input data are assumed to be nonnegative integer numbers. Classical decision-making problems arising in this context have been coarsely classified in [1] as follows:

Q1: Is it possible to arrange all jobs on a fixed set of $r \in \mathbb{N}$ given servers?

Q2: Which is the subset of jobs that yields the largest profit when arranged on $r \in \mathbb{N}$ given servers?

Q3: Which is the smallest number $r \in \mathbb{N}$ of servers needed to arrange all jobs?

Although Q1 can be interpreted as a decision version of Q3, and thus there are strong relationships between these two questions, the relevant literature has initially focussed intensively on Q2. In this context, the *temporal knapsack problem (TKP)*, where a profit-maximal allocation of a single server (i.e., $r = 1$) is required, is probably the most significant special case. However, referring to concrete applications in communications theory, this problem was originally introduced as the *bandwidth allocation problem (BAP)*, see [7, 14], generalizing some preliminary concepts from an earlier publication [2] in machine scheduling. In that framework, a server can also be thought of as a fixed-capacity communication channel for which there are certain requests to reserve bandwidth (that is, to transmit information). Consequently, assuming the profit of each job to be proportional to its area in the capacity-time space, its contribution to the objective function is directly linked to the amount of information conveyed by it, see [7]. Given the state of computational hardware and commercial software (for the exact solution of such problems) at that time, the literature initially focussed on heuristic methods and corresponding performance analyses, but also on complexity-theoretic aspects of the overall problem or of special cases. For the latter, important milestones can be summarized as follows:

- The TKP with *uniform weights*, i.e., a scenario where all c_i are identical, was shown to be polynomially solvable in $\mathcal{O}(n^2 \log n)$ in [2, Theorem 1] by drawing connections to the coloring of interval graphs and minimum cost flow problems, respectively[1]. In that special case, the capacity constraint reduces to a cardinality constraint, and the problem under consideration is also referred to as the *interval scheduling problem*, see [34] for a good survey article.

- In contrast, the TKP with *uniform profits*, where p_i is assumed to be identical for all jobs, turns out to be $\mathcal{NP}$-hard, see [18, Theorem 1] for a proof drawing a connection to a specific partition problem. For the sake of completeness, we mention that the TKP is called the *resource allocation problem (RAP)* in that publication.

- In [14, Theorem 1], the TKP was shown to be polynomially solvable if the capacity C is not part of the input. To this end, an $\mathcal{O}(n^{C+1})$ time algorithm based on dynamic programming was described, establishing some early foundations for a graph-theoretic interpretation of the TKP. Note that a similar observation was already part of [2, Theorem 3], but this result did not refer to the traditional TKP.

- For the general TKP, significant contributions were made in [14]. Here, the authors distinguish between concrete specifications in bandwidth or memory allocation. The main difference is that in the second application, the so-called *storage allocation problem (SAP)*, the jobs must be positioned as actual rectangles, i.e., they particularly must consume contiguous capacity units at any time. This represents a fundamental difference to the BAP, but both application examples are reminiscent of classical two-dimensional assignment problems of cutting and packing. In fact, the authors also point out strong relations to the *multidimensional knapsack problem*, see [33] for a very good overview. However, the dimension of this substitute problem highly depends on the input data of the TKP (more precisely, the time horizon), so that both problems (BAP and SAP) turn out to be $\mathcal{NP}$-hard even in the very restricted case when $s_i = 0$, $e_i = 1$, and $p_i = c_i$ hold for all jobs.

[1] In fact, the authors of [2] consider a slightly different problem description with C identical machines having unit capacity, and $c_i = 1$ for all $i \in I$. However, on closer examination, this is nothing else than a TKP on a single server with capacity C.

As can be seen from this overview, already the TKP (which did not yet appear under this name in the aforementioned publications) is a very challenging problem from a theoretical point of view. For this reason, heuristic methods have been an essential subject of further investigations. We refer the interested reader to some classic approaches proposed in [6, 14, 18] and to algorithms for handling more general problem aspects such as online scenarios with preemption [7] and specific job durations that can be smaller than the activity interval, so that there more flexibility in execution [39].

1.2. Problem-specific Literature Review

From a more mathematically-oriented perspective, the TKP was formally introduced in an article addressing an application in the context of resource allocation in high performance computing, see [8]. In that work, the authors suggested tackling the TKP by techniques combining constraint programming and branch-and-cut, but their algorithms were not as strong as a direct solution of the integer program by the commercial CPLEX-solver. Some years later, however, solution methods (for the TKP) were significantly advanced by the works on Dantzig-Wolfe decomposition methods [12, 13] . Remarkably, in that strategy, the (relatively large) set of constraints is first systematically partitioned, then each of these (relatively small) partition classes is separately convexified, and finally the partial solutions obtained in this way are harmonized. In that regard, it is of particular benefit that an arbitrary but fixed variable occurs exclusively in successive constraints, see also [27, Chapter 3]. Meanwhile, further refinements of such decomposition methods have been discussed in the literature, see [15, 29] for two recent examples. Moreover, a standard Dantzig-Wolfe decomposition also empirically proved to be very helpful for a multi-server version of TKP, referred to as the *operational interval scheduling with a resource constraint (ORSIC)* in [1]. However, as the authors admit, even this approach does not address the question Q3 from the above list, which is identified as interesting future work in the concluding section of [1], but has remained untouched (at least in terms of powerful solution approaches) for quite a long time in the scientific community.

To this end, the current paper focuses on the optimization problem pertinent to that open question, namely the *temporal bin packing problem (TBPP)*, which consists of finding the minimum number of servers required to accommodate all given jobs. Even though it is a rather obvious generalization of the well-known *bin packing problem (BPP)*, see [25, 42, 43] for some very good and thorough overviews, the TBPP has recently been mentioned for the first time in the context of a concrete practical application from computer science in [19]. In fact, the alarming predictions concerning the exponential increase in the energy consumption of physical computing resources, illustrated in recent studies like [4], have made industry and scientific communities take notice, and consequently prompted a sustained intensification of the debate and research on more energy-efficient operating options, see [11, 32] for some general aspects and [26] for some specific ideas and results of a leading European research cluster.

Given its relations to other already well-studied optimization problems (like strip packing), which exist but are less helpful in detail, see [36, Section 1], the consideration of exact solution methods for the TBPP represents an independent branch of research, see [23] for a first rigorous investigation of compact models and further more sophisticated methods. As a result of that research, currently, the most promising algorithm for solving the TBPP is a branch-and-price method that incorporates numerous lower bounds and heuristics and leads to convincing results in numerical test calculations. Despite all these efforts, still not all associated benchmark instances can be solved optimally in reasonable time, as reported in [23].

The last statement also applies, and even more clearly, to the much younger *temporal bin packing problem with fire-ups (TBPP-FU)*. In that scenario, we assume the same input data as for the TBPP, but in the objective function we have to minimize a weighted sum of the number of servers in use and the number of switch-on processes (so-called *fire-ups*) required during operation. The second objective is thereby provided with a weighting factor $\gamma > 0$. This inclusion of another criterion generally leads to integer programs of even larger size which are therefore typically more difficult to solve. For $\gamma \leq 1/n$ it was shown that a solution of the TBPP-FU always solves the TBPP as well and thus both problems are relatively close, see [3]. For other choices of the weighting parameter

- a solution with minimum number of servers (in terms of TBPP) does not have to be optimal for the TBPP-FU, see [3, Example 2.2],

- the possibility to decompose an instance (of the TBPP-FU) in a temporal sense is lost, see [38, Theorem 3],

so that an obvious relation between the two problems does not exist anymore. Although the compact models for the TBPP-FU (called M1 and M2), originally introduced in [3], have been substantially improved over the past two years [36, 37, 38], only about 66% of the problem-specific benchmark instances can be solved optimally in reasonable time. In addition, if the benchmark sets formerly designed in [23] for the traditional TBPP are now also taken into account, many more instances of moderate size cannot be tackled successfully, see [36].

Thus, for both variants of temporal bin packing considered here, good exact approaches (either compact models or branch-and-price) have been found and their properties have been optimized to a large extent, but numerous instances still remain unsolved. This article therefore proposes the concept of flow formulations, which is still (almost) entirely unexplored for both problems under consideration. Flow formulations form a powerful tool in cutting and packing, as they combine important structural properties (e.g., a good LP relaxation) with a large illustrativeness and a generally manageable model size, and so they can be handled efficiently by commercial solvers. In particular, extensions of the flow models originally described in detail for the first time in [43] have therefore been widely used in the recent past to present competitive exact approaches to fundamental optimization problems such as the cutting stock problem [21, 24], the skiving stock problem [35], or the multiple knapsack problem [22]. In particular, the importance of the general methodology is also highlighted by the recent survey article [20].

1.3. Our Contribution

While all these very successful approaches have in common that they require a (pseudo-)polynomial number of states (nodes) and transitions (arcs) and thus allow the efficient treatment as an ILP formulation, such a graph-theoretic formulation for the TBPP is not yet known and not within reach. As already described for the example of the TKP, see [2] and [12], the only way out is therefore via a graph which has an exponential number of states and transitions. Such an approach is also called a *combinatorial flow model* and is, however, according to the previous sources (and also [27, page 22]), only useful if, for example, the number of simultaneously active jobs at any point in time is very restricted – a property that is generally not given for the benchmark instances mentioned before. In addition, even the authors of a very recent work on exponential-size networks to tackle the TKP, see [17], admit that a straightforward application of the graph-theoretic idea does not lead to an efficient solution framework. Probably for these reasons, such an approach to temporal bin packing problems has not yet been investigated in the literature at all.

With this paper, we would like to contribute to foster the research on flow-based approaches to the two optimization problems under consideration. The main results of this work are the following:

- For both, the TBPP and the TBPP-FU, we present a layer-based combinatorial flow model. Here, each layer corresponds to a maximal clique of the interval graph belonging to the instance.
- Our approach is different from the previous attempts from the literature, see [12] and [17], as it uses another interpretation of states and transitions, leading to much smaller (but still exponentially large) networks.
- We improve the combinatorial flow models obtained in this way by valid inequalities.
- For both problems, the TBPP and the TBPP-FU, all known benchmark instances can be solved exactly in reasonable time. Moreover, significantly better computation times are achieved for those instances that could already be handled with the methods from the literature. As an outlook, we also try to explore the limits of our combinatorial arcflow model by dealing with instance sizes much larger than reported in [23] and [36].

We highlight that, although the paper just addresses two important application problems, which are related but have relevant structural differences, combinatorial flow models have a much wider applicability, and pave the way for further very powerful solution techniques to other interval scheduling problems discussed in the introductory parts.

2. The Temporal Bin Packing Problem: Preliminaries and Solution Methods

Let us consider a list of $n \in \mathbb{N}$ *items* (*jobs*), specified by an item *size* (*resource demand*) $c_i > 0$ and an *activity interval* (*lifespan*) $[s_i, e_i)$ with $s_i < e_i$, $i \in I := \{1, \ldots, n\}$, and a sufficiently large number of homogeneous *bins* (*servers*) of *capacity* $C > 0$. We will refer to s_i and e_i by the *starting time* and *ending time* (or *terminating time*), respectively. Without loss of generality, we make the following assumptions:

- All input data are integers.
- The items are sorted with respect to non-decreasing starting times (where ties are broken in an arbitrary way).
- The statement $c_i \leq C$ holds for all $i \in I$ (because the problem would become infeasible otherwise).

Then, the *temporal bin packing problem (TBPP)* requires to schedule the jobs to a minimum number of servers, so that the capacity of any server is respected at any instant of time. To briefly refer to a particular TBPP, we introduce the following well-known term.

Definition 1. *A tuple $E = (n, C, \boldsymbol{c}, \boldsymbol{s}, \boldsymbol{e})$, where $\boldsymbol{c}$, $\boldsymbol{s}$, and $\boldsymbol{e}$ are n-dimensional vectors collecting the input-data (size, starting time, ending time) of the items, is called an instance (of the TBPP).*

Typically, we refer to the set of time instants by $T := \bigcup_{i \in I} \{s_i, e_i\}$, and address the set of starting times by $T_S = \bigcup_{i \in I} \{s_i\}$. Moreover, the set $I_t := \{i \in I \,|\, t \in [s_i, e_i)\}$ collects all jobs that are active at time $t \in T$.

Example 1. *Let us consider the instance*

$$E_0 = (5, 5, (2, 2, 3, 2, 1), (1, 2, 5, 7, 12), (3, 14, 10, 8, 13))$$

which is displayed in Fig. 1 and taken from [27, Sect. 3.2]. In this scenario, we obviously have $T = \{1, 2, 3, 5, 7, 8, 10, 12, 13, 14\}$, $T_S = \{1, 2, 5, 7, 12\}$, and (by way of example) $I_7 = \{2, 3, 4\}$. Note that an optimal solution requires two servers.

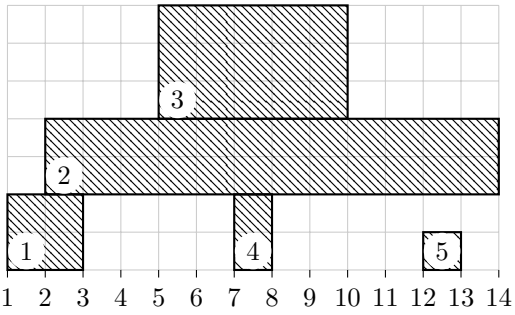

Figure 1: An illustration of the instance E_0. The horizontal axis specifies the time instants, while the vertical grid measures the item sizes.

Remember that, although the items are visualized as ordinary rectangles in the "capacity-time plane" in Fig. 1, in a feasible solution of the TBPP they can in fact also be packed in such a way that they do not represent connected objects, see Fig. 2 for an example.

Remark 1. *A similar illustration was already presented in [23, Figure 2] to show that the TBPP uses a different concept of feasible configurations than, for instance, the SPP. It should be noted that the instance depicted in Fig. 2 is smaller than the existing counterexample from the literature, both in terms of the number of items and the server capacity.*

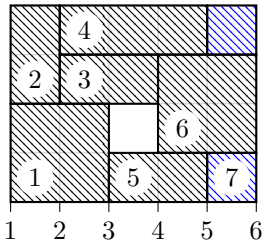

Figure 2: An assignment of seven items to one bin of size $C = 5$, following an idea from bandwidth allocation in wireless networks, see [14]. The blue item $i = 7$ with $[s_i, e_i) = [5, 6)$ and $c_i = 2$ cannot be placed in a connected manner. For instance, this packing would not be feasible for the strip packing problem (SPP).

Following the structure of Kantorovich-type models for the BPP, see [31], a first compact formulation for the TBPP was proposed in [23] and can be obtained as follows. With $K := \{1, \ldots, n\}$ denoting the set of all servers, we can introduce two types of binary variables:

- We define $z_k \in \{0, 1\}$ with $z_k = 1$ if and only if server $k \in K$ is used.
- We define $x_{ik} \in \{0, 1\}$ with $x_{ik} = 1$ if and only if item $i \in I$ is assigned to server $k \in K$.

Then, we obtain the

Compact Model for the TBPP (from [23])

$$z^{com} = \sum_{k \in K} z_k \to \min$$

$$\text{s.t.} \quad \sum_{k \in K} x_{ik} = 1, \qquad i \in I, \tag{1}$$

$$\sum_{i \in I_t} c_i x_{ik} \leq C \cdot z_k, \qquad t \in T, k \in K, \tag{2}$$

$$x_{ik} \in \{0, 1\}, \qquad i \in I, k \in K, \tag{3}$$

$$z_k \in \{0, 1\}, \qquad k \in K. \tag{4}$$

The objective function minimizes the total number of servers in use. Moreover, the two sets of constraints make sure that any job is executed precisely once (see (1)) and that the capacity of the servers is respected at any instant of time (see (2)). Additionally, the latter prevent jobs from being assigned to unused servers at all.

Obviously, for a fixed server $k \in K$, it is sufficient to require Conditions (2) only for all $t \in T_S$, since the load on a server can increase only at precisely these points in time. In fact, we can even go one step further.

Definition 2 ([23]). *Let E be an instance of the TBPP and let $t_1 < t_2 \in T_S$ follow each other directly in the chronologically ordered set T of all time instants. If t_2 is not an end time, then t_1 is dominated by t_2. The set of all non-dominated starting times is referred to as $T_S^{nd} \subseteq T_S$.*

In a situation like the one described in the definition, all jobs that are active at t_1 are still active at t_2, meaning that the associated capacity condition (2) for $t = t_2$ contains all the terms that would appear in the constraint for $t = t_1$, so that it dominates that restriction for all $k \in K$. Hence, Constraints (2) only need to be formulated for the non-dominated starting times.

Example 2. *For our toy instance E_0, illustrated in Fig. 1, we conclude that $T_S^{nd} = \{2, 7, 12\}$. By way of example, the former element $t_1 = 1 \in T_S$ is dominated by $t_2 = 2$.*

Despite these possible improvements, such assignment-based ILP models usually have two major drawbacks, which are also evident here:

- The set of feasible solutions is highly symmetric due to permutations of the server indices.
- The LP bound coincides with a problem-specific generalization of the material bound, so that it is typically rather poor, see [23, Property 1].

Due to these disadvantageous properties of the compact model, a pattern-based approach has been established in the literature, see [23], with a structure strongly reminiscent of the Gilmore-Gomory model of one-dimensional cutting stock problems, see [28].

Definition 3. *Any feasible assignment of jobs to a single server is called a pattern.*

Mathematically, a pattern can be described by an n-dimensional incidence vector $\boldsymbol{x} \in \{0,1\}^n$ (or, equivalently, as a subset $U \subseteq I$) the components of which contain the information whether item $i \in I$ is contained in the pattern or not. Hence, the set of patterns for the TBPP is given by

$$\mathcal{P} := \mathcal{P}(E) := \left\{ \boldsymbol{x} \in \{0,1\}^n \;\middle|\; \sum_{i \in I_t} c_i x_i \leq C,\, t \in T \right\},$$

where T can also be replaced by T_S^{nd}. Due to the numerous combination possibilities, the cardinality of this pattern set typically grows exponentially with the number n of jobs appearing in an instance. Let $\mathcal{J}$ denote an index set of $\mathcal{P}$, then we can introduce a decision variable $\xi_j \in \{0,1\}$ for each pattern $j \in \mathcal{J}$, stating whether it is used ($\xi_j = 1$) or not ($\xi_j = 0$). By that, we obtain the

Exponential-size Model for the TBPP (from [23])

$$z^{exp} = \sum_{j \in \mathcal{J}} \xi_j \to \min$$

$$\text{s.t.} \quad \sum_{j \in \mathcal{J}} \xi_j x_i^j = 1, \qquad i \in I, \tag{5}$$

$$\xi_j \in \{0,1\}, \qquad j \in \mathcal{J}. \tag{6}$$

Again, the total number of servers is minimized while ensuring that any job is contained in precisely one pattern used, see Constraints (5). The exponential-size formulation does not contain any symmetry, and also its LP bound is generally better than that of the compact model presented before, see [23, Property 4]. Since the LP relaxation of the exponential-size model can be solved efficiently by column generation, the currently best solution approach for the TBPP, called B&P$^+$, uses a branch-and-price algorithm based on that formulation, see [23, Sect. 6]. Before starting the actual (and costly) branch-and-price main procedure, the algorithm first tries to solve a given instance exactly using various lower bounds and heuristics.

- Determining appropriate lower bounds is mainly done by computing the rounded-up LP values $\lceil z_{LP}^{exp,\star} \rceil$.
- During the first phases of the algorithm, the previously determined lower bounds are compared with a plethora of heuristic values. The heuristics used for this comparison are sorted by ascending difficulty and complexity. Thus, first an attempt is made to prove optimality for a given instance using very simple heuristics (e.g., first-fit techniques for the original and the lifted instance, see also [10, 16] for the general concept of *lifting*), before moving to successively more sophisticated procedures culminating in token-based *diving heuristics*. The latter were proposed and discussed intensively in [41] and intend to descend within a small part of the branch-and-bound tree according to a fixed (simple) heuristic rule until a suitable integer feasible solution is obtained. The value of the token thereby regulates that the numerically more difficult branching path $\xi_j = 0$ (i.e., the decision not to use a certain pattern) can be chosen only very rarely. In contrast, the simpler branching path $\xi_j = 1$ (which allows continuing with a reduced and thus easier TBPP by removing the items occurring in the chosen pattern) may be used as often as desired.

Only when these previous techniques could not yet find a proven optimal solution for the given instance, the actual branch-and-price procedure is started. Here, the authors deviate from the classical pattern-based branching scheme, i.e. $\xi_j = 0$ vs. $\xi_j = 1$. Instead, the algorithm uses a branching rule due to Ryan and Foster, see [40], a strategy that considers several variables at once per node and, thus, typically offers rather balanced branching trees and a more efficient performance of the overall algorithm. For corresponding justifications and further explanations of the incorporation of the additional conditions into the respective subproblems, we refer to the general explanations in [5, Sect. 4] as well as the problem-specific contributions to the TBPP in [23, Sect. 6].

Altogether, the overall state-of-the-art algorithm B&P$^+$ is able to solve to proven optimality TBPP instances with up to 500 items and 150 non-dominated starting times, in reasonable computing times, as literally reported in [23]. However, not all the benchmark instances can be solved to proven optimality yet.

3. A Combinatorial Flow-based Formulation for the Temporal Bin Packing Problem

Although there is no graph-theoretical formulation for the TBPP in the relevant literature so far, two main concepts for the underlying TKP could constitute a starting point for further considerations. These approaches each describe layer-based graphs of exponential size, but they differ significantly in how such a layer is constructed. More specifically, the details of these two frameworks are given as follows:

(I) In [12, Subsection 3.3], a *clique-based* idea already partly outlined in [2] is discussed for the TKP. In that approach, the number of layers in the graph is determined by the number of maximal cliques of the interval graph belonging to the given instance. We note that there is a one-to-one relationship between the non-dominated starting times, see Definition 2, and the maximal cliques, implying a natural order among the maximal cliques. Moreover, the latter can be efficiently determined in polynomial time, see [9] or [27, Algorithm 1] for an implementation with $\mathcal{O}(n^2)$ time. Let us define $\mathcal{C}_0 := \emptyset$ and $\mathcal{V}_0 := \{\emptyset\}$ to represent an artificial first layer. For any of the remaining layers, consider a fixed maximal clique $\mathcal{C}_l$. Then, the basic idea used in [12] is to define the states (nodes) $\mathcal{V}_l$ occurring in layer l of the graph as all feasible server allocations that can be built with the items of $\mathcal{C}_l$, i.e., we have

$$\mathcal{V}_l := \left\{ J \subseteq \mathcal{C}_l \,:\, \sum_{i \in J} c_i \leq C \right\}.$$

In other words, $\mathcal{V}_l$ somewhat collects the "subpatterns" relevant for clique $\mathcal{C}_l$. For any layer index $l \geq 1$, the arc set $\mathcal{E}_l$ between layer $l-1$ and layer l is defined as follows:

$$\left(\underline{J}, \overline{J}\right) \in \mathcal{E}_l \iff \underline{J} \in \mathcal{V}_{l-1},\, \overline{J} \in \mathcal{V}_l,\, \underline{J} \cap \mathcal{C}_l = \overline{J} \cap \mathcal{C}_{l-1}.$$

This definition particularly implies that both states, $\underline{J}$ and $\overline{J}$, have to contain the same items from the set $\mathcal{C}_{l-1} \cap \mathcal{C}_l$, so that, among others, the artificial source node in $\mathcal{C}_0$ is connected to any node of the first layer. It is straightforward to see that there are at most $\mathcal{O}(2^{|\mathcal{C}_l|})$ nodes in layer l, so that the overall graph has an exponential size.

(II) In [17], on the other hand, an *event-based* approach is presented that leads to a graph whose number of layers is, by and large, determined by twice the number of items. Each of these layers belongs to a particular event, i.e., either the start or the completion of a job. In addition, there is an artificial last layer (with index $2n+1$), which consists only of a dummy sink node. In the graph itself, a state is described by a triple $(e, c, \boldsymbol{a})$, where e specifies the event and c is the capacity consumed by pattern $\boldsymbol{a} \in \mathcal{P}$. Of course, it is important to note that $\boldsymbol{a}$ can only use those items that are consistent with the event under consideration. A transition from one state to the other can occur exactly when the associated item is picked up (at its start time) or released (at its end time). In the first case, it must also be checked whether the capacity condition is still fulfilled, while the second case requires the completed item to be part of the server state before. It is clear that, also for this variant, the state space will grow exponentially with the number of items.

As we have seen, both approaches generally lead to very large graphs, which result, among other things, from focusing on the pattern set and/or having to represent each item twice (that is, by two events) during graph generation. However, the latter, as well as a certain degree of redundancy in the representation of nodes, is necessary for the application of the relaxation techniques presented in [17].

Therefore, in the following we would like to describe a possibility which, on the one hand, avoids redundant information in the labels and, on the other hand, leads to significantly smaller graphs, although we also put the patterns themselves in the center of our construction. In turn, however, our graph will sometimes have multiple arcs between two nodes. Let us start with two motivating examples to draw a direct connection to the previous attempts from the relevant literature, before going through the precise construction.

Example 3. *We consider again the instance E_0 from Example 1 and follow the idea of [12]. Obviously, the instance has three maximal cliques, namely $\mathcal{C}_1 = \{1,2\}$, $\mathcal{C}_2 = \{2,3,4\}$, and $\mathcal{C}_3 = \{2,5\}$, so the traditional concept presented in [12] will lead to a graph consisting of four layers (three for the cliques and one dummy layer for the source node). For example, the first actual layer $l = 1$ contains the states $\mathcal{V}_1 = \{\emptyset, \{1\}, \{2\}, \{1,2\}\}$, while $\mathcal{V}_2$ and $\mathcal{V}_3$ consist of seven and four elements, respectively. Thus, in total, the graph has 16 nodes, which are connected by 32 arcs. Given its relatively large size we omit an illustration here, but note that one can be found in [27, Figure 3.3]. Looking at this graph a bit more closely, we see, for example, that the nodes $\emptyset$ and $\{1\}$ from layer $l = 1$ are connected to exactly the same four nodes from layer $l = 2$ (namely, $\emptyset$, $\{3\}$, $\{4\}$, and $\{3,4\}$). Thus, from the point of view of the arcs emanating from layer $l = 1$, these two states are to be evaluated as equivalent. This is also not unexpected, because at the end of the time interval relevant for clique $\mathcal{C}_1$, job $i = 1$ has already ended, so that it is irrelevant for the further server utilization (in the subsequent layers) which of the states $\emptyset$ or $\{1\}$ was once selected in layer $l = 1$. Hence, we suggest equipping a clique-based layer only with one representative per set of equivalent states (in the sense described before). Of course, then we have to attach the information which items were actually chosen to some other component of the graph. For that purpose, we will finally add a corresponding label to the arcs and allow multiple arcs between the states of two consecutive layers, if required. In Fig. 3, the combinatorial arcflow graph for E_0 is depicted. Although the precise construction details have not yet been revealed, we see that it contains the same four clique-based layers, but only a total number of six nodes (illustrated as rectangles with rounded corners) and 15 arcs (black lines with rectangular label placed in their center). So, in fact, the arcs of our approach somewhat carry the information of the states appearing in [27, Fig. 3.3], so that both, the number of nodes and arcs in Fig. 3, is much smaller than before.*

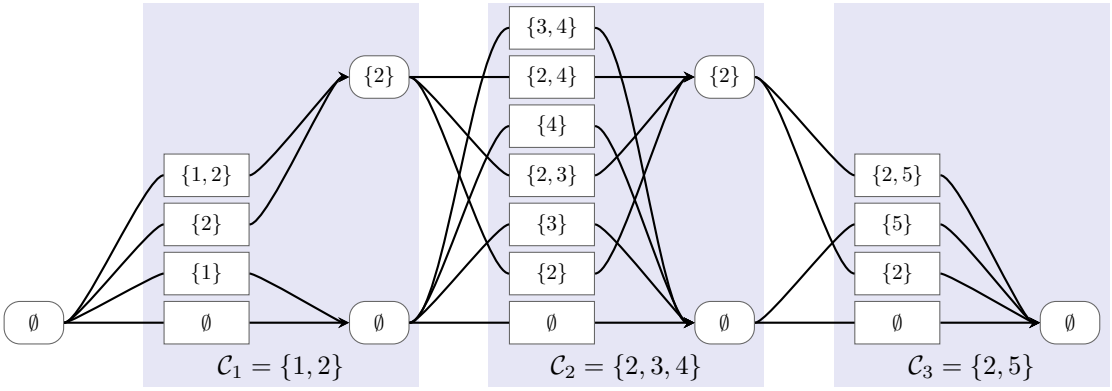

Figure 3: The combinatorial arcflow graph for E_0 consisting of four layers, six nodes, and 15 arcs. In contrast to the idea presented in [12], the arc and node labels carry some information relevant to the respective clique (so that, in our visualization, both of them are related to the blue background indicating a specific maximal clique).

Example 4. *We highlight that, for the toy instance $E_1 := (3, 8, (4,4,8), (0,1,3), (2,4,5))$ appearing in [17], our idea would lead to a graph having three layers (the two maximal cliques are $\mathcal{C}_1 = \{1,2\}$ and*

$\mathcal{C}_2 = \{2, 3\}$*), four nodes, and seven arcs, see Fig. 4. In contrast, the event-based approach also results in a much larger network consisting of seven layers,* 15 *nodes, and* 18 *arcs, see [17, Figure 2].*

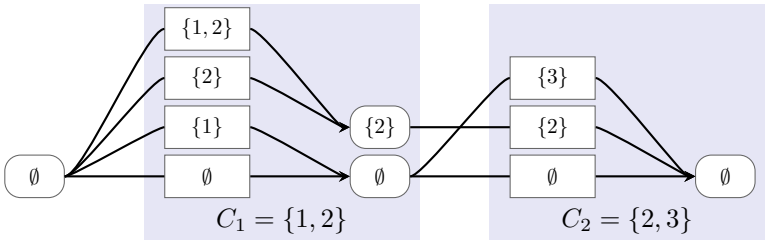

Figure 4: The combinatorial arcflow graph for the instance E_1 taken from [17].

Already from these examples we can see that our approach will lead to a much more efficient representation of the graphs. Although, from a theoretical point of view, they still are exponential in size, we will observe later in the numerical test calculations (see Sect. 5) that both the generation of these graphs and the direct application of an ILP solver to the flow problem belonging to it does typically not consume an unreasonable amount of time anymore. In other words, for the first time our construction makes the exponential size controllable without requiring additional techniques.

We would now like to formalize the idea outlined in the previous examples. To this end, let E be a fixed instance of the TBPP, and let $\boldsymbol{\mathcal{C}} := \{\mathcal{C}_1, \dots, \mathcal{C}_m\}$ denote the set of its maximal cliques sorted in increasing order with respect to the associated non-dominated starting times. Moreover, we define two index sets $L := \{1, \dots, m\}$ and $L_0 := \{0\} \cup L$ to refer to the cliques and the layers, respectively. Lastly, remember that $\mathcal{P}$ represents the set of all feasible patterns of E. In a slight abuse of notation, when describing and visualizing the ideas of our graph construction we will usually not refer to patterns by their incidence vectors. Instead, to not display too many redundant zero entries, we will make use of the corresponding subsets $J \subseteq I$. Since there is an obvious one-to-one relation between these two concepts, no harm will arise from statements like $J \in \mathcal{P}$.

In our construction of the directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, any node will be referred to as a pair (l, J) with $l \in L_0$ and some $J \subseteq I$. So, the first entry contains information about the layer, whereas the second specifies a subset of the items. Similarly, we will define an arc by a tuple $(l-1, l, J)$ with $l \in L$ and some $J \subseteq I$, thus describing a transition from layer $l-1$ to l caused by J. Note that, whenever an arc is concerned, the subset J appearing in the tuple $(l-1, l, J)$ can also be interpreted as a (sub)pattern from

$$\mathcal{P}_l := \left\{ U \subseteq \mathcal{C}_l \,:\, \sum_{i \in U} c_i \leq C \right\}.$$

To initialize the set of nodes, let us define the dummy layer $\mathcal{C}_0$ with node set $\mathcal{V}_0 := \{(0, \emptyset)\}$. For any $l \geq 1$, we define (in two equivalent ways)

$$\mathcal{V}_l := \{(l, U \cap \mathcal{C}_l \cap \mathcal{C}_{l+1}) \,:\, U \in \mathcal{P}\} = \{(l, U \cap \mathcal{C}_{l+1}) \,:\, U \in \mathcal{P}_l\}.$$

For the special case $l = m$ (that is, the final layer), we use $\mathcal{C}_{m+1} := \emptyset$ in the above definition. So, in fact, $\mathcal{V}_l$ collects the possible server states at time

$$e(l) := \max\{e_i \,:\, i \in \mathcal{C}_l \setminus \mathcal{C}_{l+1}\},$$

that can be observed if only the items exclusive to the set of cliques $\mathcal{C}_1, \dots, \mathcal{C}_l$ processed so far are considered. In other words, these are the representative states that have been described in Example 3. Note that, due to the second possibility to define $\mathcal{V}_l$, we typically do not have to cope with the complete pattern set, but only with $\mathcal{P}_l$, so that the total number of vertices does not grow as fast as in the approaches

known from the literature.

Let $\mathcal{E}_l$, $l \in L$, denote the set of arcs from layer $l-1$ to layer l. Similar (but not identical) to the approach presented in [12] and [27], there is some arc between two nodes $(l-1, \underline{J}) \in \mathcal{V}_{l-1}$ and $(l, \overline{J}) \in \mathcal{V}_l$ if and only if $\underline{J} \cap \mathcal{C}_{l+1} = \overline{J} \cap \mathcal{C}_{l-1}$. However, for our network, we have to be more precise, because we can have multiple arcs between the same pair of nodes. So, the previous definition, in fact, just tells us that there is *at least* one arc between the two specific nodes, but further information is still missing. Hence, we demand that the nodes $(l-1, \underline{J}) \in \mathcal{V}_{l-1}$ and $(l, \overline{J}) \in \mathcal{V}_l$ are connected if and only if

$$\exists J \in \mathcal{P}_l : J \cap \mathcal{C}_{l-1} = \underline{J},\ J \cap \mathcal{C}_{l+1} = \overline{J}$$

holds, and we have to draw a separate arc for any such J. Hence, associated to each arc in $\mathcal{E}_l$, there is a unique subset of I (in fact, a subpattern from $\mathcal{P}_l$), called J in the previous definition, so that we can use this set to label the arc. As a consequence, we are allowed to abstractly refer to an arc by a tuple $(l-1, l, J)$ with some $J \in \mathcal{P}_l$, so that the notation introduced earlier is justified.

Remark 2. *Note that any pattern $J \in \mathcal{P}$ corresponds to a unique directed path from the source node $(0, \emptyset)$ to the sink node $(m, \emptyset)$. Indeed, that path has to use the vertices $(l, J \cap \mathcal{C}_l \cap \mathcal{C}_{l+1})$, $l \in L_0$, and the arcs $(l-1, l, J \cap \mathcal{C}_l)$, $l \in L$. On the other hand, each path connecting the source and the sink node of the network via the arcs $(0, 1, J_1)$, $(1, 2, J_2)$, ..., $(m-1, m, J_m)$ defines a unique pattern, namely $J = J_1 \cup \ldots \cup J_m$ (or, more accurately, the corresponding incidence vector).*

To conveniently formulate an integer optimization problem, let us collect all arcs referring to the positioning of item $i \in I$ in the set $\mathcal{E}(i)$. In other terms, we define

$$\mathcal{E}(i) := \{(l-1, l, J) \in \mathcal{E} \,:\, i \in J \setminus \mathcal{C}_{l-1}, l \in L\}.$$

Moreover, the arcs entering and leaving a given state $(l, J) \in \mathcal{V}$ will be denoted by $\mathcal{E}^{in}(l, J)$ and $\mathcal{E}^{out}(l, J)$, respectively. Now, let us introduce an integer variable $\xi_{l-1,l,J} \in \mathbb{Z}_+$ representing the units of flow carried by an arc $(l-1, l, J) \in \mathcal{E}_l$, $l \in L$. For the sake of simplicity, we will always use e to abbreviate the elements contained in a specific set of arcs. Then, we obtain the

Combinatorial Arcflow Model for the TBPP

$$z^{comb} = \sum_{e \in \mathcal{E}^{out}(0,\emptyset)} \xi_e \to \min$$

$$\text{s.t.} \quad \sum_{e \in \mathcal{E}^{in}(l,\widetilde{J})} \xi_e = \sum_{e \in \mathcal{E}^{out}(l,\widetilde{J})} \xi_e, \qquad (l, \widetilde{J}) \in \mathcal{V} \setminus \{(0,\emptyset), (m,\emptyset)\}, \tag{7}$$

$$\sum_{e \in \mathcal{E}(i)} \xi_e = 1, \qquad i \in I, \tag{8}$$

$$\xi_e \in \mathbb{Z}_+, \qquad e \in \mathcal{E}_l, l \in L. \tag{9}$$

Obviously, the objective function minimizes the total flow in the network (that is, the number of required servers), while Constraints (7) ensure the flow conservation at every vertex (except for the source and the sink node). Moreover, Conditions (8) manage that every job is executed precisely once. Note that, it is sufficient to restrict the flow to zero or one for most of the arcs. However, there are some arcs (namely, the arcs $(l-1, l, \emptyset)$, $l \in L$, representing a transition from one empty state to the next by means of $J = \emptyset$) which can be used by multiple patterns. So, for exactly those arcs it is necessary to have integer-valued flow variables. To simplify the presentation of the model, here we do not differentiate between these two possibilities.

Now that we have presented and thoroughly explained the admittedly technical details of our graph construction, we would like to conclude by briefly discussing another example. In contrast to the previous ones at the beginning of the current section, this one was not taken from the literature, but is already designed in such a way that we can then continue to work with it in the following section (when fire-ups have to be taken into account, too).

Example 5. *Let us consider the instance* $E_2 := (5, 5, (1, 2, 5, 2, 4), (1, 2, 5, 6, 9), (7, 10, 6, 9, 10))$*, see Fig. A.10 in the appendix for a graphical illustration. Obviously, the maximal cliques are given by* $\mathcal{C}_1 = \{1, 2, 3\}$*,* $\mathcal{C}_2 = \{1, 2, 4\}$*, and* $\mathcal{C}_3 = \{2, 5\}$*, so that our network will consist of four layers, eight nodes, and* 16 *arcs, see Fig. 5. For the sake of completeness, in the appendix we also provide the graphs resulting from the approaches of [17] and [12] in Fig. A.11 and A.12, respectively.*

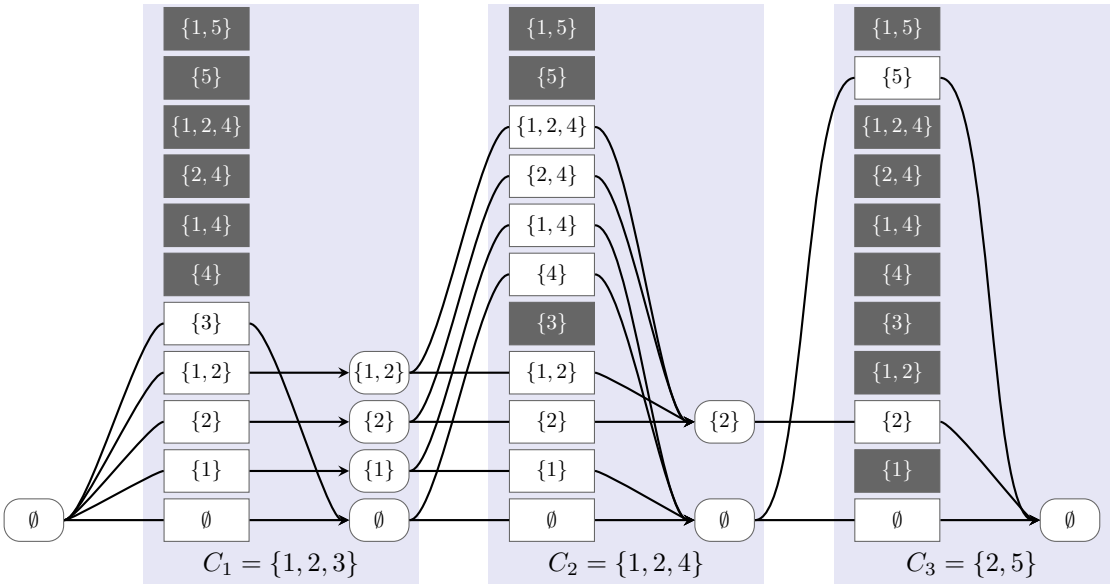

Figure 5: The combinatorial arcflow graph for E_2 from Example 5. For the sake of completeness and to better understand the overall construction process, we depicted the entire pattern set $\mathcal{P}$ in any layer, but cancelled out the elements that are infeasible because they do not use the items allowed for the respective clique. Those infeasible subpatterns are colored black and do not belong to any arc.

.

4. An Extension to the Temporal Bin Packing Problem with Fire-Ups

4.1. Preliminaries and a Basic Solution Approach

Minimizing the number of fire-ups and servers required is a very new aspect in the context of energy-efficient job-to-server scheduling introduced in [3] as the *temporal bin packing problem with fire-ups (TBPP-FU)*. The idea behind this is that not only the infrastructure as such but also its operating mode contributes to the energy consumption of the overall system. Consequently, this optimization problem assumes that servers that are temporarily unused can be put into a sleep mode or can be completely deactivated to save energy. However, such a server must then be switched on again later, at the cost of a so-called *fire-up*. Both objectives are usually addressed by a weighted sum, scaling the number of fire-ups by some parameter $\gamma > 0$, in the objective function. This approach is also justifiable from a practical point of view, since both criteria more or less describe an energy consumption, and we can therefore bundle these objectives in a joint objective function.

Definition 4. *A tuple* $E = (n, C, \boldsymbol{c}, \boldsymbol{s}, \boldsymbol{e}, \gamma)$ *where (in addition to the already known objects)* $\gamma > 0$ *represents a scaling parameter is called an instance (of the TBPP-FU).*

Although the problem description itself may not have changed much compared to the TBPP, we again point out that there are significant differences between the two considered optimization problems in terms of the solution set, provided that the scaling parameter γ is not too small, see Sect. 1.

So far, the TBPP-FU has only been studied with respect to compact formulations. Two of these (called

M1 and M2) were already suggested in [3], the article introducing this new optimization problem. Due to some obvious drawbacks of these formulations, they were subject of numerous improvements over the past two years, see [36, 37], and finally arrived at a level where no further reductions are in sight. The currently best formulation in the literature, see [38], has evolved from the original model M1, which requires four types of variables. In addition to the classic assignment-based variables (that is, z_k and x_{ik}) already known from Sect. 2, we introduce the following two sets of variables to better access the temporal aspects of the problem under consideration:

- The decision variables $y_{tk} \in \{0,1\}$ will be interpreted in the sense that $y_{tk} = 1$ represents a positive load on server k at time t.
- We use binary variables $w_{tk} \in \{0,1\}$ to state whether server $k \in K$ was activated at time $t \in T_S$.

Then, we obtain the

Assignment Model 1 (M1, original version from [3])

$$z^{(1)} = \gamma \cdot \sum_{k \in K} \sum_{t \in T_S} w_{tk} + \sum_{k \in K} z_k \to \min$$

s.t.

$$y_{tk} \leq \sum_{i \in I_t} c_i \cdot x_{ik} \leq y_{tk} \cdot C, \qquad k \in K, t \in T, \tag{10}$$

$$\sum_{k \in K} x_{ik} = 1, \qquad i \in I, \tag{11}$$

$$x_{ik} \leq y_{s_i,k}, \qquad i \in I, k \in K, \tag{12}$$

$$y_{tk} \leq z_k, \qquad k \in K, t \in T, \tag{13}$$

$$y_{tk} - y_{t-1,k} \leq w_{tk}, \qquad k \in K, t \in T_S, \tag{14}$$

$$x_{ik} \in \{0,1\}, \qquad i \in I, k \in K, \tag{15}$$

$$y_{tk} \in \{0,1\}, \qquad k \in K, t \in T, \tag{16}$$

$$w_{tk} \geq 0, \qquad k \in K, t \in T_S, \tag{17}$$

$$z_k \in \{0,1\}, \qquad k \in K. \tag{18}$$

The objective function collects the number of fire-ups (first sum) and the number of servers (second sum) and has to be minimized. While Conditions (11) already appeared in the TBPP scenario, Constraints (10) manage that the capacity is respected whenever the considered server is active at the moment. In addition, Restrictions (12)-(14) are responsible for linking the different variable types consistently. Without going further into details, we mention that, meanwhile, this original version of M1 was improved by several general and problem-specific techniques like symmetry reduction, lifting, valid cuts, and heuristic-based information, see [36, 37, 38].

4.2. A Combinatorial Flow Formulation for the TBPP-FU

As we have learned in the previous subsection, in the TBPP-FU a fire-up occurs if a server is used for the first time or if it is reactivated after it became inactive at some point back in time. So, basically, a fire-up should be registered whenever a server leaves the empty state. However, we cannot simply use the network introduced for the TBPP, because the new main challenge for our graph-theoretic approach is that no fire-up is necessary, if another jobs starts exactly at the time when a server intends to get empty. This means that it is necessary to differentiate between two possible empty states to count fire-ups correctly. More precisely, our construction will be based on a *true empty state* (shutdown), called $\emptyset$, and an *artificial empty state* (possible immediate resumption of activity), called $\emptyset_A$.

Before discussing the necessary changes in our graph construction, we have to identify in which situation the introduction of an additional empty state (in a specific layer) is mandatory. To this end, note that a server can only go into the artificial empty state if the last active job (on that server) runs exactly until

another new job starts. Otherwise, the server translates into the true empty state. To define the states accurately, we introduce the following notation for any clique $\mathcal{C}_l$, $l \in L$:

$$\begin{aligned} s(l) &:= \min\{s_i : i \in \mathcal{C}_l \setminus \mathcal{C}_{l-1}\}, \\ e(l) &:= \max\{e_i : i \in \mathcal{C}_l \setminus \mathcal{C}_{l+1}\}. \end{aligned}$$

Note that the second one already briefly appeared in Sect. 3, but we think mentioning it again will help to remember its meaning within the following constructions. From a descriptive point of view, $s(l)$ is the earliest starting time of jobs that are introduced in $\mathcal{C}_l$, whereas $e(l)$ refers to the latest ending time of jobs that are completed in $\mathcal{C}_l$. This means that a node representing the artificial empty state has to exist in layer l, $l \in L \setminus \{m\}$, if and only if $e(l) = s(l+1)$ holds.

Example 6. *For the instance E_2 from our previous example, we obtain the arcflow graph depicted in Fig. 6, if the TBPP-FU is considered.*

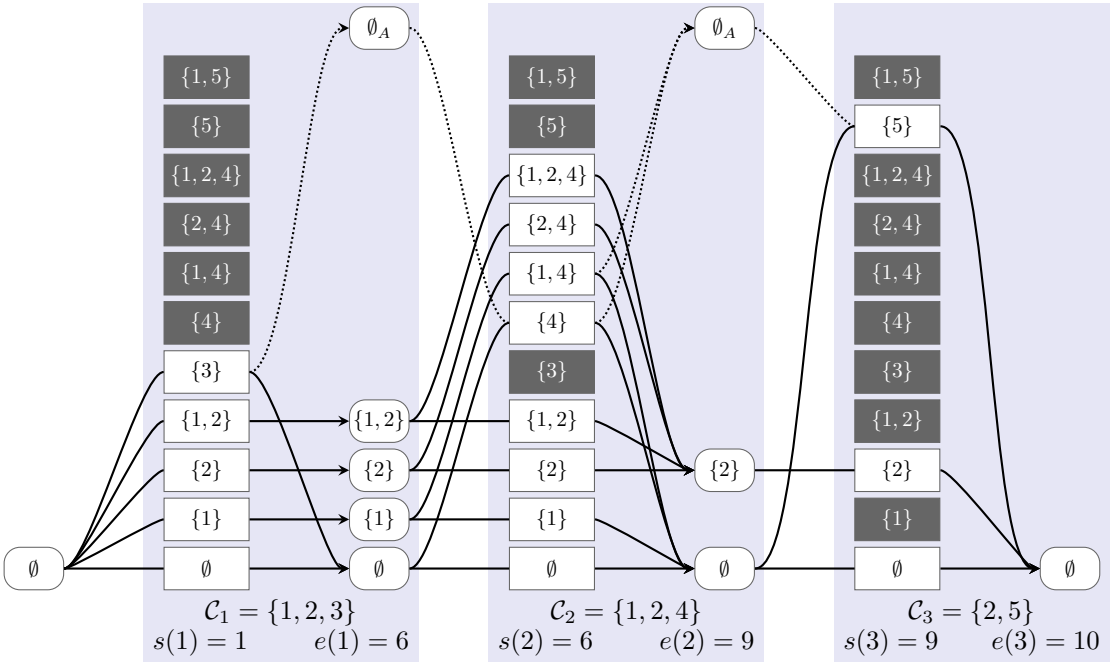

Figure 6: The combinatorial arcflow graph for the TBPP-FU and the instance E_2.

Even if the essential differences to the graph generation of the TBPP have already been summarized, we would now like to discuss the formal definitions of the node and arc sets in more detail. To better distinguish between standard and artificial objects, we introduce additional tags in the description for both the states and the transitions. More precisely, we let $\mathcal{V}_l^S$, $l \in L_0$, and $\mathcal{V}^S$ denote the set of standard nodes, i.e., these are the vertices which already appeared in the TBPP graph from Sect. 3. In addition, we define

$$\mathcal{V}^A := \{(l, \emptyset_A) : l \in L \setminus \{m\}, e(l) = s(l+1)\}$$

to refer to the artificial empty states, whenever they are required in the respective layer of the graph. Of course, the set of all nodes is then given by $\mathcal{V} := \mathcal{V}^S \cup \mathcal{V}^A$.

To correctly reconstruct the fire-ups later, it is convenient to attach an additional fourth component to the description of an arc. First of all, we again have a set $\mathcal{E}^S$ of standard arcs between two non-artificial states which can be inherited from the TBPP graph, see Sect. 3. However, any such arc is now referred to

as a 4-tuple $(l-1, l, J, S)$, where the tag 'S' refers to 'standard'. Similarly, any arc involving an artificial state can be assigned to one of the following three groups:

$$\begin{aligned}
\mathcal{E}^{A\leftarrow} &:= \left\{(l-1, l, J, \leftarrow) \,:\, l \in L,\ J \in \mathcal{P}_l,\ \max_{i\in J}\{e_i\} = s(l+1)\right\},\\
\mathcal{E}^{A\rightarrow} &:= \left\{(l-1, l, J, \rightarrow) \,:\, l \in L,\ J \in \mathcal{P}_l,\ \min_{i\in J}\{s_i\} = e(l-1)\right\},\\
\mathcal{E}^{A\leftrightarrow} &:= \left\{(l-1, l, J, \leftrightarrow) \,:\, (l-1, l, J, \rightarrow) \in \mathcal{E}^{A\rightarrow} \text{ and } (l-1, l, J, \leftarrow) \in \mathcal{E}^{A\leftarrow}\right\}.
\end{aligned}$$

We define the set of all arcs as $\mathcal{E} := \mathcal{E}^S \cup \mathcal{E}^{A\leftarrow} \cup \mathcal{E}^{A\rightarrow} \cup \mathcal{E}^{A\leftrightarrow}$. The new types of arcs represent the following transitions.

- An arc $(l-1, l, J, \leftarrow) \in \mathcal{E}^{A\leftarrow}$ connects a state in $\mathcal{V}^S_{l-1}$ (namely, the state belonging to $J \cap \mathcal{C}_{l-1}$) to the subsequent artificial empty state $(l, \emptyset_A) \in \mathcal{V}^A$. This corresponds to a server which would get empty at time $s(l+1)$, but could continue execution if a suitable job is assigned.
- Similarly, an arc $(l-1, l, J, \rightarrow) \in \mathcal{E}^{A\rightarrow}$ starts in the artificial empty state $(l-1, \emptyset_A) \in \mathcal{V}^A$ and goes to a state in $\mathcal{V}^S_l$ (namely, the state belonging to $J \cap \mathcal{C}_l$). In this transition, a server which got empty recently is directly reactivated (without requiring a fire-up).
- An arc $(l-1, l, J, \leftrightarrow) \in \mathcal{E}^{A\leftrightarrow}$ connects one artificial empty state $(l-1, \emptyset_A) \in \mathcal{V}^A$ to the subsequent artificial empty state $(l, \emptyset_A) \in \mathcal{V}^A$ (via the subpattern $J \in \mathcal{P}_l$). In fact, this transition is a combination of the previous ones.

Remark 3. *For the instance E_2 dealt with in Sect. 3, we obtain the following sets of non-standard arcs*

$$\begin{aligned}
\mathcal{E}^{A\leftarrow} &= \{(0, 1, \{3\}, \leftarrow), (1, 2, \{1, 4\}, \leftarrow), (1, 2, \{4\}, \leftarrow)\},\\
\mathcal{E}^{A\rightarrow} &= \{(1, 2, \{4\}, \rightarrow), (2, 3, \{5\}, \rightarrow)\},\\
\mathcal{E}^{A\leftrightarrow} &= \{(1, 2, \{4\}, \leftrightarrow)\},
\end{aligned}$$

which can also be found in Fig. 6. Whenever their "trajectory" is not covered by another standard arc, we displayed them by dotted lines. Note that extending the representation of arcs to 4-tuples is indeed necessary, because in our example there are four different scenarios to move from layer $l = 1$ to layer $l = 2$ via the subpattern $J = \{4\}$, but the arcs $(1, 2, \{4\}, \rightarrow), (1, 2, \{4\}, \leftrightarrow)$ do not contribute to a fire-up.

A potential issue with our definition of the states is that there is no longer a one-to-one relationship between patterns (i.e., the elements of $\mathcal{P}$) and directed paths in the graph. Of course, as in the TBPP case, we still have that each path from the source to the sink corresponds to a feasible pattern, but there may be different paths leading to the same pattern, in general. More precisely, this is the case if the set $\mathcal{V}^A$ is nonempty. In particular, for any path which goes through some artificial empty state $(l, \emptyset_A) \in \mathcal{V}^A$, there is another path using the node $(l, \emptyset) \in \mathcal{V}^S$ instead and ends up with the same pattern, see Fig. A.13 in the appendix for an example. The necessary arcs for this replacement must exist by the definition of the graph. As regards our optimization model, however, this ambiguity is typically not problematic since a path through an artificial empty state is preferred over the equivalent path using a true empty state because of the fire-up costs $\gamma > 0$.

For the purpose of a preferably convenient modeling, we again let $\mathcal{E}(i)$ denote the set of all arcs which represents the starting of job $i \in I$. Moreover, with κ acting as a generic tag symbol, we specify the incoming and outgoing arcs of some node $(l, \widetilde{J}) \in \mathcal{V}$ as follows:

$$\mathcal{E}^{in}(l, \widetilde{J}) := \begin{cases} \left\{(l-1, l, J, \kappa) \in \mathcal{E} \,:\, J \cap \mathcal{C}_l = \widetilde{J}, \kappa \in \{S, \rightarrow\}\right\} & \text{if } (l, \widetilde{J}) \in \mathcal{V}^S,\\ \{(l-1, l, J, \kappa) \in \mathcal{E} \,:\, \kappa \in \{\leftarrow, \leftrightarrow\}\} & \text{if } (l, \widetilde{J}) = (l, \emptyset_A) \in \mathcal{V}^A. \end{cases}$$

$$\mathcal{E}^{out}(l, \widetilde{J}) := \begin{cases} \left\{(l, l+1, J, \kappa) \in \mathcal{E} \,:\, J \cap \mathcal{C}_{l+1} = \widetilde{J}, \kappa \in \{S, \leftarrow\}\right\} & \text{if } (l, \widetilde{J}) \in \mathcal{V}^S,\\ \{(l, l+1, J, \kappa) \in \mathcal{E} \,:\, \kappa \in \{\rightarrow, \leftrightarrow\}\} & \text{if } (l, \widetilde{J}) = (l, \emptyset_A) \in \mathcal{V}^A. \end{cases}$$

The main difference to the previous combinatorial flow model is that we have to correctly perceive the fire-ups. To this end, we make use of set

$$\mathcal{E}^{FU} := \{(l-1, l, J, \kappa) \in \mathcal{E} \,:\, l \in L, J \cap \mathcal{C}_{l-1} = \emptyset, J \cap \mathcal{C}_l \neq \emptyset, \kappa \in \{S, \leftarrow\}\}\,.$$

In other words, a fire-up has to be counted when connecting the true empty state with a non-empty state via a standard arc (with tag 'S') or, alternatively, with an artificial empty state via an arc from $\mathcal{E}^{A\leftarrow}$. Note that precisely those two constellations relate to a time period without any load on the server, whereas in addition the first possibility also takes care of recognizing the first activation of a server.

With all these ingredients at hand, we can introduce integer variables $\xi_{l-1,l,J,\kappa} \in \mathbb{Z}_+$ denoting the units of flow carried by arc $(l-1, l, J, \kappa) \in \mathcal{E}_l$, $l \in L$. Then, we obtain the

Combinatorial Arcflow Model for the TBPP-FU

$$z^{comb} = \sum_{e \in \mathcal{E}^{out}(0,\emptyset)} \xi_e + \gamma \cdot \sum_{e \in \mathcal{E}^{FU}} \xi_e \to \min$$

$$\text{s.t.} \quad \sum_{e \in \mathcal{E}^{in}(l,\widetilde{J})} \xi_e = \sum_{e \in \mathcal{E}^{out}(l,\widetilde{J})} \xi_e, \qquad (l, \widetilde{J}) \in \mathcal{V} \setminus \{(0,\emptyset), (m,\emptyset)\}, \tag{19}$$

$$\sum_{e \in \mathcal{E}(i)} \xi_e = 1, \qquad i \in I, \tag{20}$$

$$\xi_e \in \mathbb{Z}_+, \qquad e = (l-1, l, J, \kappa) \in \mathcal{E}_l, l \in L. \tag{21}$$

In fact, the general form and the interpretation of the constraints did not change when moving from the TBPP to the TBPP-FU. The only considerable difference is that the objective function now also contains a sum collecting the fire-up terms.

5. Numerical Tests

5.1. Computational Environment and Test Instances

In the literature, the following benchmark sets (referred to as *Category A* and *Category B*) have been described for the two problems under consideration:

(A) In [3, Sect. 5], the authors suggested 48 differently characterized groups of 5 instances each, forming a set of 240 instances in total. All instances share the values $C = 100$, $\gamma = 1$, but any two groups differ in precisely one of the criteria:

- number of items: $n \in \{50, 100, 150, 200, 500, 1000\}$,
- time horizon: dense scenario ($\bar{s} := \max_{i \in I}\{s_i\} = n$) vs. relaxed scenario ($\bar{s} = 1.2n$),
- job duration: short ('d_S') vs. long ('d_L'),
- capacity consumption: low ('c_L') vs. high ('c_H').

For the precise construction, we refer the interested reader to the aforementioned publication. From the input data, we can see that there is a wide range of possible values, especially with respect to the number of jobs, and these can be used to further decompose (A) into two subcategories:

(A1) In this subset, we would like to summarize those 160 instances with values $n \leq 200$. Those instances have been tackled and (partially) solved in [3] using exact approaches, and they are also used in the articles dealing with improved compact formulations, see [36, 37].

(A2) In this subset, we would like to gather the 80 significantly more difficult instances with $n \in \{500, 1000\}$. These have been treated in the literature so far exclusively with heuristic methods, see [3], so no information about optimal solutions is available.

(B) In [23, Sect. 7] the authors introduced a set of 1500 instances with $C = 100$, originating from an earlier investigation of the TKP in [12]. In contrast to Category (A), the range of the item sizes is not that restricted, and the main input parameter is given by the number of non-dominated starting times (maximal cliques). More precisely, for any $|T_S^{nd}| \in \{10, 20, 30, \ldots, 150\}$ a set of 100 instances (divided into ten classes called I-X) is considered. Any class is described by a parameter a denoting the average number of items per clique, and a parameter b influencing the job duration. So, the higher the a-parameter or the lower the b-parameter of an instance, the more jobs will have to be assigned in total. The full details of that construction can be found in [23]. Here, we just highlight that Classes VI and IX possess relatively small b-parameters, whereas Classes VIII-X exhibit relatively large values of the a-parameter, so these classes contain the more challenging instances.

Remark 4. *For the sake of completeness, it should be noted that Category (B) also contains two sets of* 100 *instances each with* $|T_S^{nd}| \in \{5, 15\}$. *However, according to the experiments conducted in [23], we will not make use of these (relatively easy) instance sets.*

In the following subsections, we compare our combinatorial arcflow approach with the best known solution methods from the literature. To recapitulate, we have the B&P$^+$ algorithm from [23] for the TBPP and a compact model for the TBPP-FU (originating from the formulation M1) presented in [38]. The new flow-based approach is coded in Python (version 3.10.1) and solved by Gurobi (version 9.5) on an AMD A10-5800K processor with 16 GB RAM. Unless stated otherwise, we use a time limit of 30 minutes. However, especially for some very large instances appearing in Subsect. 5.5, we will also perform computations without any time limit.

5.2. Structural Comparison of the Graph-based Approaches

Before dealing with the concrete numerical performance of our new approach, we would like to study its general applicability in more detail. This is done in particular against the background of allowing a comparison to the approaches from the literature, but also to show that the exponential size of the network, which still exists (in theory), now appears to be much more controllable. To this end, we collected the average size of the three graphs (in terms of nodes and arcs) for two representative instance sets from Category (B) in Tab. 1.

Remark 5. *To give some more information about the instances, we also list the average size of the cliques* $|\mathcal{C}_l|_{avg}$ *(i.e., the items per clique) and the average total number of items* $|I|_{avg}$. *Note that, effectively, the first value is identical to the mean value of the a-parameter used to construct the instance class in Category (B), see [23] for the details.*

We clearly see that combinatorial arcflow (termed 'CAF') is much smaller than the two competitors from the literature. Having a look at the average numbers of states and transitions, there are remarkable savings of between 85 and 90% compared to the event-based graph presented in [17]. A similar, but not to the identical degree superior, result is obtained in comparison with the layer-based idea from [12]. Here, the reduction in terms of nodes is slightly above 37%, whereas the number of arcs decreases by almost 60%.

Remark 6. *An interesting side aspect is given by the observation that, as a direct consequence of the construction itself, the number of nodes in the approach from [12] is almost equal to the number of arcs in CAF. In fact, only the source node of [12] does not appear as an arc in our implementation.*

To also obtain a rough impression of the numerical data with respect to other parameter values of $|T_S^{nd}|$ or the specific class index, in addition to the representative numbers in Tab. 1, we display the normalized arc numbers (i.e., the value $|\mathcal{E}|$ of CAF is scaled to 1) of Category (B) for the approaches from [12] and [17] in Fig. 7. This is of particular interest because the number of arcs is identical to the number of integer variables in the ILP model and, thus, significantly determines the solution efforts. First, it is noticeable that after initial fluctuations, a relatively constant ratio is obtained for increasing values of $|T_S^{nd}|$. We attribute this to the fact that the construction of a fixed instance class, independent of $|T_S^{nd}|$ itself, always follows the same principle, and, therefore, basic structural properties (like the "density" of the arcs between the layers) of the graphs are preserved for each of the three approaches. It can be further

$\lvert T_S^{nd}\rvert$	Class	$\lvert\mathcal{C}_l\rvert_{avg}$	$\lvert I\rvert_{avg}$	number of nodes ($\lvert\mathcal{V}\rvert$)			number of arcs ($\lvert\mathcal{E}\rvert$)		
				[17]	[12]	CAF	[17]	[12]	CAF
50	I	10.0	59.0	4.7	2.5	**1.9**	5.2	3.2	**2.4**
	II	15.0	97.0	20.1	6.7	**4.9**	22.0	10.1	**6.7**
	III	20.0	110.0	33.3	9.8	**7.5**	35.6	14.0	**9.8**
	IV	25.0	139.1	113.0	26.9	**19.6**	120.4	41.6	**26.9**
	V	30.0	161.2	214.1	45.0	**32.6**	226.5	70.0	**45.0**
	VI	30.0	339.7	424.7	48.8	**22.2**	451.4	208.6	**48.8**
	VII	30.0	161.8	209.2	42.9	**31.4**	220.7	66.9	**42.9**
	VIII	30.0	213.5	330.0	49.6	**30.4**	349.3	93.9	**49.6**
	IX	29.9	354.4	442.4	46.8	**20.9**	468.3	208.7	**46.8**
	X	34.9	231.3	436.4	62.3	**40.8**	457.9	114.2	**62.3**
	Average	25.5	186.7	222.8	34.1	**21.2**	235.7	83.1	**34.1**
100	I	10.0	109.0	8.5	4.6	**3.7**	9.5	6.1	**4.6**
	II	15.0	181.0	39.0	13.4	**9.8**	42.7	20.5	**13.4**
	III	20.0	201.3	61.6	18.6	**14.4**	65.9	26.6	**18.6**
	IV	25.0	255.1	209.4	51.0	**37.2**	223.2	79.1	**51.0**
	V	30.0	294.0	354.2	75.3	**54.8**	374.8	118.5	**75.3**
	VI	30.0	657.1	760.8	87.8	**40.7**	808.0	375.2	**87.8**
	VII	30.0	295.6	371.4	78.1	**57.4**	392.1	122.5	**78.1**
	VIII	30.0	402.5	630.3	96.7	**60.5**	666.6	181.8	**96.7**
	IX	30.1	689.9	872.8	92.1	**41.1**	923.9	422.3	**92.1**
	X	35.0	432.3	982.4	140.7	**91.2**	1032.0	258.0	**140.7**
	Average	25.5	351.8	429.0	65.8	**41.1**	453.9	161.0	**65.8**

Table 1: The average number of nodes and arcs (in units of 10^3) for three different arcflow graphs: the event-based version from [17], the layer-based variant from [12], and combinatorial arcflow ('CAF') presented in this work. For the sake of exposition, we just consider two representative sets of instances from Category (B).

seen that even for values of $|T_S^{nd}|$ other than those studied in Tab. 1, CAF achieves a large saving over the networks from the literature, especially compared to [17]. The effect varies by class and is particularly pronounced, then also in comparison with [12], for Classes VI and IX, which have the most items (see Tab. 1) and are, thus, particularly challenging.

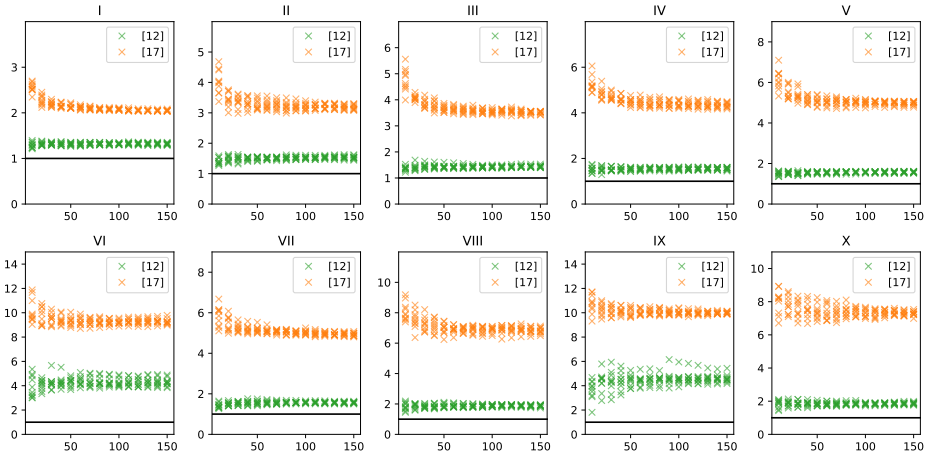

Figure 7: Number of arcs (normalized to CAF) for the networks presented in [12] and [17] for the ten classes belonging to Category (B). The horizontal axis represents the parameter $|T_S^{nd}|$ of the instances.

The new approach has by far the most promising structural properties, so there is a great confidence to be able to disprove the previous statements from the literature, according to which a direct application of ILP solvers to such a combinatorial graph is not an efficient solution method. This line of reasoning

is supported by the fact that we see particularly large improvements in the number of arcs, i.e., the subsequent model variables, and thus arrive at potentially less challenging ILP formulations.

However, all these expectations are pointless if the generation of the still exponentially large network alone takes too much time to be able to solve it subsequently. To reject this possible criticism as well, we list the average times to generate the graph (including the corresponding ILP model for the TBPP) in Tab. 2. Given this data and the time limit of 30 minutes, it can be stated that the construction of the graph generally requires only a relatively small amount of time, even for larger instances, and thus the ILP model can be easily passed to a solver. We also point out that with other programming languages, even shorter runtimes could be expected for model generation, because the performance of statically typed, compiled languages is typically much better (compared to Python).

$\lvert T_S^{nd}\rvert$	10	20	30	40	50	60	70	80	90	100	110	120	130	140	150
I	0.0	0.1	0.1	0.2	0.2	0.2	0.3	0.3	0.4	0.4	0.5	0.5	0.5	0.5	0.6
II	0.1	0.2	0.3	0.5	0.6	0.7	0.8	1.0	1.1	1.2	1.3	1.4	1.6	1.7	1.9
III	0.1	0.3	0.5	0.7	0.9	1.0	1.2	1.3	1.6	1.7	1.9	2.2	2.6	2.8	3.1
IV	0.5	1.0	1.3	2.0	2.5	3.0	3.4	3.8	4.3	4.8	5.4	5.8	6.4	6.6	7.0
V	0.7	1.4	2.3	3.5	4.3	4.8	5.4	6.0	6.5	7.1	8.2	9.1	9.9	10.7	11.4
VI	0.7	1.3	2.1	2.8	3.7	4.4	5.0	5.6	6.3	6.9	7.7	8.7	9.3	10.2	11.0
VII	1.0	1.8	2.6	3.3	4.1	4.7	5.5	6.2	6.9	7.5	8.5	9.2	10.0	10.9	11.9
VIII	0.7	1.5	2.2	3.3	4.3	5.3	6.5	7.2	8.0	8.7	9.8	10.3	11.1	12.2	12.8
IX	0.5	1.3	2.2	2.8	3.6	4.3	4.9	5.5	6.4	7.2	7.9	8.8	9.3	9.8	10.5
X	1.1	2.1	3.4	4.5	6.0	7.2	8.7	10.1	11.2	13.0	14.5	15.9	17.7	20.1	21.9
Average	0.6	1.1	1.7	2.3	3.0	3.6	4.2	4.7	5.3	5.8	6.6	7.2	7.8	8.6	9.2

Table 2: The average time (in seconds) to construct the CAF network (and the associated ILP).

Remark 7. *It is interesting to note that it does not matter which of the networks (TBPP or TBPP-FU) is considered. The times to generate the graphs (and the corresponding ILP formulation) are very close to each other, because there is only a tiny difference with respect to the number of nodes and arcs between the two frameworks. Since this argument should be clear from the constructions described in the previous sections, in Tab. 1 and Tab. 2 we do not intend to provide additional data for the case of the TBPP-FU.*

As a conclusion of this subsection, we would like to summarize that our graph is, in fact, much smaller than the previous approaches from the literature, and its generation takes only a relatively short time (especially measured against the available time limit). The graph we present is therefore very well suited to be used for an exact solution by ILP solvers. The corresponding results obtained from numerical computations involving the benchmark sets presented before will be documented and discussed in the following subsections.

5.3. Numerical Results for the TBPP

As mentioned in Sect. 2, currently the best method for solving TBPP instances is the B&P$^+$ algorithm developed in [23]. This approach has already been able to solve the vast majority of the 1500 TBPP-specific benchmark instances (that is, Category (B)) in reasonable time. However, there are still 29 of these instances – all of which having $|T_S^{nd}| \geq 90$ and originating exclusively from the Classes V-X – where no proven optimal solution was found. To this end, in Tab. 3 and Tab. 4 we copied the results of the previous state of the literature (so, the running times and number of optimal solutions reported in [23]), and compare them with the performance of CAF. We note that in [23] a time limit of one hour was used.

We see that CAF can now solve every single instance, typically requiring significantly less computation time in each of the subsets of instances considered in the tables. Looking at the averages over all instances (in the last row), we observe a reduction in computation time of almost 90%, which on closer inspection, for example, becomes even larger for particularly challenging choices of the parameter $|T_S^{nd}|$, see Tab. 3. On the other hand, we also note that for some of the more difficult instance classes, such as Classes VI, VIII, and X, there are still much more remarkable performance gains of up to roughly 95%, see Tab. 4.

By way of example, for Class VI we already saw in Fig. 7 that, compared to the large number of items, CAF leads to a particularly small graph representation, so that the convincing performance noticed in Tab. 4 could have been expected.

	CAF with $t_{\max} = 1800s$		[23] with $t_{\max} = 3600s$	
$\lvert T_S^{nd} \rvert$	t	opt	t	opt
10	**0.4**	**(100)**	2.0	(100)
20	**1.4**	**(100)**	5.2	(100)
30	**2.5**	**(100)**	9.3	(100)
40	**3.7**	**(100)**	15.7	(100)
50	**6.9**	**(100)**	27.9	(100)
60	**10.0**	**(100)**	63.9	(100)
70	**15.7**	**(100)**	115.1	(100)
80	**16.5**	**(100)**	132.6	(100)
90	**20.4**	**(100)**	151.8	(99)
100	**23.9**	**(100)**	168.8	(99)
110	**26.7**	**(100)**	218.0	(98)
120	**30.7**	**(100)**	237.9	(99)
130	**38.6**	**(100)**	432.9	(94)
140	**40.2**	**(100)**	475.1	(92)
150	**51.5**	**(100)**	559.4	(90)
Average	**19.3**	**(1500)**	174.4	(1471)

Table 3: Numerical comparison (for the TBPP) between CAF and the best solution approach from the literature, that is, B&P$^+$ from [23], for Category (B) (ordered by number of non-dominated starting times).

	CAF with $t_{\max} = 1800s$		[23] with $t_{\max} = 3600s$	
Class	t	opt	t	opt
I	**0.2**	**(150)**	0.6	(150)
II	**2.0**	**(150)**	9.4	(150)
III	**1.8**	**(150)**	8.1	(150)
IV	**25.3**	**(150)**	60.1	(150)
V	**49.0**	**(150)**	396.0	(146)
VI	**11.1**	**(150)**	214.7	(146)
VII	**48.9**	**(150)**	248.7	(147)
VIII	**17.7**	**(150)**	223.9	(144)
IX	**10.6**	**(150)**	82.3	(147)
X	**26.2**	**(150)**	500.3	(141)
Average	**19.3**	**(1500)**	174.4	(1471)

Table 4: Numerical comparison (for the TBPP) between CAF and the best solution approach from the literature, that is, B&P$^+$ from [23], for Category (B) (ordered by instance classes).

Remark 8. *We note that the tabulated computation times for CAF are only the pure solution times of the ILP solver. However, this does not distort the previous statements in any way, as we have seen in Tab. 2 how small the modeling times turn out to be despite the exponential size. So, even if we added these times to the average values of the solution time, we would still observe a clear victory of CAF for each considered subset.*

A somewhat more detailed overview of both solution methods is shown in a performance profile, see Fig. 8. This illustration displays the percentage of optimally solved instances over time, and it reveals that, apart from a small interval around a computation time of one second, CAF is strictly better than B&P$^+$ at any point in time.

5.4. Numerical Results for the TBPP-FU

Since we are now able to solve any known benchmark instance for the TBPP in a short time with the help of our new approach, we would now like to turn to the somewhat more complex TBPP-FU. At the

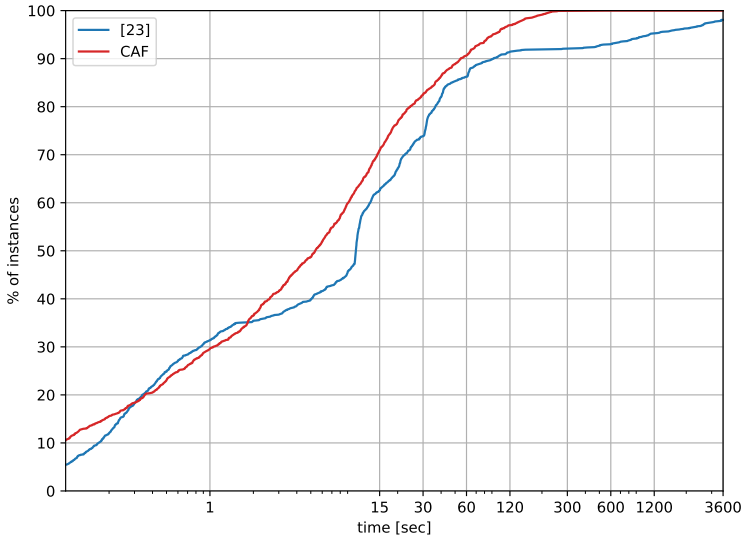

Figure 8: Performance profile for the comparison of CAF and B&P$^+$ from [23] applied to Category (B).

beginning, we note that we have added the lower bound $h := \lceil z_{LP}^{exp,\star} \rceil$ for the number of servers in use to the model from Sect. 4 by requiring

$$\sum_{e \in \mathcal{E}^{out}(0,\emptyset)} \xi_e \geq h \qquad \text{and} \qquad \sum_{e \in \mathcal{E}^{FU}} \xi_e \geq h. \tag{22}$$

Observe that this bound can be obtained in relatively short time by solving the LP relaxation of the ordinary TBPP, and therefore it already appeared in any of the compact models on the TBPP-FU proposed in the literature[2]. Despite numerous improvements of these compact formulations, only about two thirds of the 160 classical test instances from Category (A1) could be solved to proven optimality so far. Currently, the most successful approach is a model of M1-type, see Sect. 4, whose final variant was recently presented in [38].

Remark 9. *For the sake of completeness, we mention that the latest Gurobi version 9.5 was not yet available for the calculations conducted in [38]. However, as we will see in Tab. 5, the performance gains observed when switching to the flow-based approach are so impressive that they cannot result just from a slightly updated software package alone.*

The results listed in Tab. 5 contain the following main information:

- CAF can solve any single benchmark instance from Category (A1). Moreover, the solution times are (almost) always much better than reported in [38]. Having a look at the overall average, we see that the solution time has reduced by more than 90% again.
- Since we have so far collected only exemplary model generation times for instances of Category (B), Tab. 5 also contains the respective average values t_{mod} for the instances considered here. It is again noticeable that these generation times are much smaller than the available time limit (and, thus, acceptable). Except for some very easy instance classes with a rather small number of items, where the time required for both formulations is in the very low seconds range, CAF wins

[2]To be more precise, the lower bound on the number of servers (that is, the first inequality from (22)) directly appears in any compact formulation from the literature, whereas the lower bound on the number of fire-ups (that is, the second inequality from (22)) is implicitly imposed by means of valid inequalities, see [36]. The important overall message, then, is that CAF has no advantage based on modeling by adding the lower bounds presented in (22).

				CAF			[38]	
n	$\bar{s}$	d_i	c_i	t_{mod}	t	opt	t	opt
50	50	d_S	c_L	1.0	1.6	(5)	4.0	(5)
			c_H	0.2	0.1	(5)	1.3	(5)
		d_L	c_L	1.7	0.8	(5)	360.8	(4)
			c_H	0.5	0.1	(5)	0.5	(5)
	60	d_S	c_L	0.6	0.9	(5)	1.2	(5)
			c_H	0.3	0.2	(5)	1.7	(5)
		d_L	c_L	2.0	1.2	(5)	11.2	(5)
			c_H	0.5	0.1	(5)	0.6	(5)
Average (Sum)				0.9	**0.6**	**(40)**	47.7	(39)
100	100	d_S	c_L	2.1	4.2	(5)	3.6	(5)
			c_H	0.7	0.3	(5)	78.6	(5)
		d_L	c_L	10.9	78.2	(5)	1449.8	(1)
			c_H	3.0	1.7	(5)	1092.9	(2)
	120	d_S	c_L	1.3	3.1	(5)	85.3	(5)
			c_H	0.5	0.3	(5)	83.6	(5)
		d_L	c_L	5.6	13.7	(5)	685.9	(4)
			c_H	1.7	0.7	(5)	546.9	(4)
Average (Sum)				3.2	**12.8**	**(40)**	503.3	(31)
150	150	d_S	c_L	3.2	13.2	(5)	85.7	(5)
			c_H	1.0	0.6	(5)	1464.1	(1)
		d_L	c_L	18.9	177.1	(5)	1462.2	(1)
			c_H	4.2	4.2	(5)	1372.0	(2)
	180	d_S	c_L	2.1	8.0	(5)	37.2	(5)
			c_H	0.9	0.4	(5)	853.9	(3)
		d_L	c_L	13.0	175.1	(5)	1198.4	(3)
			c_H	3.0	2.2	(5)	1494.0	(1)
Average (Sum)				5.8	**47.6**	**(40)**	995.9	(21)
200	200	d_S	c_L	5.1	47.1	(5)	99.2	(5)
			c_H	1.5	1.6	(5)	1800.0	(0)
		d_L	c_L	29.9	571.0	(5)	1800.0	(0)
			c_H	7.3	9.1	(5)	1624.6	(1)
	240	d_S	c_L	3.3	14.1	(5)	123.1	(5)
			c_H	1.2	0.7	(5)	1693.9	(2)
		d_L	c_L	18.6	283.2	(5)	1201.9	(2)
			c_H	5.0	14.2	(5)	1800.0	(0)
Average (Sum)				9.0	**117.6**	**(40)**	1267.8	(15)
Total: Average (Sum)				4.7	**44.7**	**(160)**	703.7	(106)

Table 5: Numerical comparison (for the TBPP-FU) between CAF and the best solution approach from the literature, that is, a compact M1-type model from [38], for Category (A1). In addition to the pure solution times, we also report about t_{mod}, the time to build the network and the corresponding ILP formulation.

the comparison with the compact formulation (generally clearly) even when the modeling times are taken into account.

- For some of the (d_S, c_H) scenarios, we see that $t_{mod} \approx t$ or even $t_{mod} > t$ holds for CAF. However, as can be observed from the comparison with t_{mod} of other parameter configurations, this is not an indication of a disproportionately large modeling time. In fact, in these scenarios, the graph has many cliques of rather small cardinality (since the job durations are short), together with relatively few feasible patterns (since many items are incompatible) and, consequently, a somewhat small number of arcs. Thus, the optimization problems obtained for CAF generally have the fewest variables and can therefore be solved particularly fast. We will come back to this point with more details later in Tab. 8.

- In particular, those instances where there are many possible item combinations (i.e., the constellation (d_L, c_L)) proved to be very difficult or even intractable for all the compact formulations, see [37]. In

contrast, CAF can also solve the hardest subsets of these instances to proven optimality in less than ten minutes on average. However, we clearly see, especially when compared to the computation times of the ordinary TBPP reported in Tab. 3, that these are indeed already somewhat more challenging instances even for CAF. This is because, with such a large number of possible temporal interactions between the items, the maximal cliques generally consist of many different items and, thus, the number of nodes (as well as, implicitly, the number of arcs) increases considerably compared to other parameter constellations. Therefore, the modeling time is by far the largest for these instance groups, and the relatively large flow-based ILP to be solved then also requires some noticeable computation times.

Typically, another advantage of flow models over compact formulations is given by a substantially better LP bound, see [20]. Here, too, for Category (A1) a corresponding dominance relation can be manifested empirically in Tab. 6. However, except for the constellation (d_S, c_H) being the most favorable setup in terms of a small network, the deviations are typically less than 3% and, therefore, not as considerable as one might have expected. On the one hand, this is due to the fact that, given a second optimization criterion (and the additional model ingredients to describe it), the compact models do not possess a pure Kantorovich-type structure, but even more importantly, we have to consider that these formulations have been significantly improved by numerous techniques. In particular, any ILP model from the literature already uses the lower bound of the exponential-size TBPP formulation, which is very powerful especially in those scenarios where an optimal solution contains a small number of servers and only a few additional fire-ups (like in the setting (d_S, c_L), see also Tab. 5). Moreover, the compact ILP models have been enriched by many valid inequalities for the different types of variables, see [36, 37]. Without these techniques, the lead of CAF in terms of the LP bound would be much more convincing.

		$n = 50$		$n = 100$		$n = 150$		$n = 200$	
d_i	c_i	CAF	[38]	CAF	[38]	CAF	[38]	CAF	[38]
d_S	c_L	18.2	18.2	21.2	21.2	20.6	20.6	23.0	23.0
	c_H	24.9	24.6	32.7	31.0	36.7	34.6	39.0	35.1
d_L	c_L	29.8	29.8	32.6	32.6	35.2	35.2	35.2	35.2
	c_H	43.8	43.8	47.5	46.9	51.5	50.2	50.9	49.9
Average		**29.2**	29.1	**33.5**	32.9	**36.0**	35.2	**37.0**	35.8

Table 6: Average rounded-up LP bound for instances of Category (A1), averaged over the input parameter 'time horizon'. Hence, in this table, every number is the average of ten instances.

Remark 10. *As a consequence of the very powerful model improvements (for the compact formulations from the literature) discussed above, we note that the LP bound of CAF does not dominate the bound of the compact formulation from [38] for any possible instance, so a general theoretical result cannot be established.*

As a summary of the discussion of Category (A1) and in the light of the graphical illustration of computational results chosen in [36, 37], we would also like to provide the following performance profile, in addition to the values appearing in Tab. 5. In Fig. 9, it can be seen that CAF is clearly ahead of the best compact formulation during the entire observation period. Remarkably, after only 15 seconds almost 80 % of all instances are already solved, also making CAF suitable for applications where decisions have to be made within rather short time.

In a last experiment, we interpret the instances of Category (B) as TBPP-FU instances, to enrich the variety of test sets for the latter problem, and collect the numerical results in Tab. 7. Although there are no corresponding calculations for the compact model from [38] reported in the literature, we added some results for moderate instance sizes to enable a rough comparison. As can be seen in Tab. 7, neither of these instances is challenging for the TBPP-FU when using CAF, because (on average) they all can be solved in less than one minute, while even the harder subsets just require roughly the double amount of time. In particular, already for the 600 representative instances considered, CAF clearly outperforms the

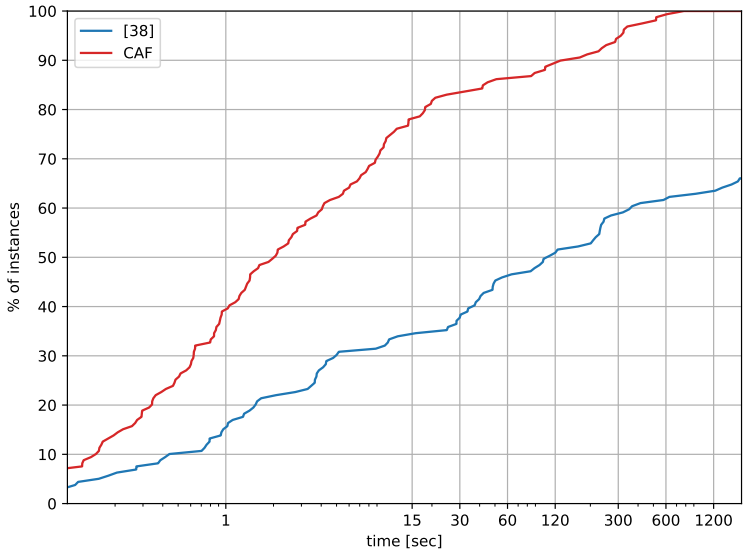

Figure 9: Performance profile for the comparison of CAF and the compact model from [38] applied to Category (A1).

results which can be obtained for the compact model from [38], so that spending more computation time to fill in the associated columns completely is definitely not required.

	TBPP		TBPP-FU		[38]	
$\lvert T_S^{nd}\rvert$	t	opt	t	opt	t	opt
10	0.4	(100)	0.5	(100)	20.0	(99)
20	1.4	(100)	1.5	(100)	144.2	(94)
30	2.5	(100)	3.9	(100)	293.0	(90)
40	3.7	(100)	12.3	(100)	461.1	(82)
50	6.9	(100)	11.4	(100)	572.4	(75)
60	10.0	(100)	17.1	(100)	684.5	(67)
70	15.7	(100)	26.8	(100)	–	–
80	16.5	(100)	46.8	(100)	–	–
90	20.4	(100)	42.8	(100)	–	–
100	23.9	(100)	53.1	(100)	–	–
110	26.7	(100)	59.4	(100)	–	–
120	30.7	(100)	70.9	(100)	–	–
130	38.6	(100)	83.9	(100)	–	–
140	40.2	(100)	97.7	(100)	–	–
150	51.5	(100)	106.7	(100)	–	–
Average (Sum)	19.3	(1500)	42.3	(1500)	–	–

Table 7: Numerical results for instances of Category (B), always with $t_{\max} = 1800s$. For the sake of an easier comparison, we repeat the solution times obtained for the classical TBPP from Tab. 3 in the first two columns of the main table. Moreover, we also display a selection of the results which would have been obtained with the approach from [38], but also point out that these results did not appear in the literature before. Due to this reason, we just conducted and included a reasonable subset of these additional calculations which is, however, fully sufficient to anticipate the general trends.

An interesting side aspect of the results presented in Tab. 7 is that, for any $|T_S^{nd}|$, solving the TBPP-FU is more time-consuming than coping with the traditional TBPP. Since neither the associated networks nor the sets of variables and constraints differ very much (among the TBPP and the TBPP-FU), we

partly attribute this to the observation that, as discussed earlier in Sect. 4, the TBPP-FU graph itself offers some symmetries to represent a feasible pattern. However, the more important reason is that any feasible schedule is now assessed by two terms (number of servers, number of fire-ups), typically entailing a wider variety of possible "numerical states" of the objective function, especially when there are much more fire-ups than active servers in an optimal solution.

5.5. Exploring the Limits: Numerical Results for Very Large Instances

In the previous subsection, we have impressively demonstrated that CAF is able to solve all benchmark instances from the literature (that have been investigated so far in the context of exact approaches) in a short time, both for the TBPP and for the TBPP-FU. We therefore provide some more numerical tests, in particular to also show the limitations of our new approach:

- For Category (A), such instances have already been collected in the literature, see [3], but they have only been treated heuristically so far. To this end, these 80 instances already appeared in the presentation at the beginning of Sect. 4 as *Category (A2)*.
- For Category (B), the relevant literature does not yet specify larger instances. Nevertheless, such instances can easily be obtained according to the known construction principles even for values $|T_S^{nd}| > 150$. To this end, applying the procedure described in [23] to the raw data from [12], we obtain more instances along the lines of Category (B). To be more precise, we thus generated 100 instances each for any parameter value $|T_S^{nd}| \in \{160, 170, \ldots, 400\}$. Although there is no set (B1), we will refer to these 2500 very large instances as *Category (B2)* to synchronize with the terminology of Category (A).

Let us proceed in alphabetical order. Before examining the actual numerical results of Category (A2), we would first like to present some data on the size of the respective CAF networks. For this purpose, we have summarized the number of nodes and arcs in Tab. 8, but we included all instances from Category (A) to provide a better overview of the overall evolution.

	d_i	c_i	$n = 50$	$n = 100$	$n = 150$	$n = 200$	$n = 500$	$n = 1000$
$\|\mathcal{V}\|$	d_S	c_L	5.7	11.4	17.5	27.5	67.2	140.7
		c_H	1.6	4.0	6.8	9.7	25.5	50.5
	d_L	c_L	13.1	59.3	115.5	175.3	469.4	904.9
		c_H	3.4	16.9	25.9	44.1	122.6	240.1
$\|\mathcal{E}\|$	d_S	c_L	7.7	15.8	24.2	37.8	91.4	191.5
		c_H	2.2	5.3	8.8	12.7	33.4	65.7
	d_L	c_L	17.6	71.4	137.3	206.5	553.1	1064.6
		c_H	4.4	20.1	30.2	51.4	141.5	276.4

Table 8: Average numbers of states and transitions (in units of 10^3) for the instances of Category (A). The averages are calculated based on ten instances each (since the criterion 'time horizon' is not specified here to keep the list short).

In particular, the following interesting insights should be noted:

- We see that the different parameter constellations lead to very heterogeneous graph sizes and thus cover a reasonable range of different benchmark scenarios. In particular, it becomes clear that the configuration (d_S, c_H) indeed leads to the smallest networks, while (d_L, c_L) typically allows for very many item interactions and thus requires many states and transitions. This is consistent with the associated observations of modeling and solution times (see Tab. 5), which have already been partially addressed in the related discussion before.
- Compared to Category (B), we see that the instances from Category (A) can be judged as more challenging on average. To illustrate this more thoroughly, we consider the case of $n = 500$ items as an example. Then the data from Tab. 8 prove that an associated CAF graph in the case of

Category (A) has on average about 171 thousand nodes and 205 thousand arcs. If we now search Tab. 1 for instances from Category (B) with comparable or even larger item numbers (i.e., for example, $|T_S^{nd}| = 100$ and Classes VI or X), we notice that the associated networks are generally much smaller. It is therefore to be expected that CAF will faster reach its limits in the case of Category (A), in particular since already for $n = 1000$ items partly more than one million integer variables have to be dealt with according to Tab. 8.

In view of these remarks, it is not surprising that a time limit of 30 minutes is no longer sufficient in some cases to cope with these very large instances. We have therefore also performed a calculation without any time limit in Tab. 9 to determine the so far unknown optimal value of these instances on the one hand, and to get a more precise impression of how long it actually takes to solve such challenging instances on the other hand.

					$t_{\max} = 1800s$		$t_{\max} = \infty$	
n	$\bar{s}$	d_i	c_i	t_{mod}	t	opt	t	opt
500	500	d_S	c_L	11.9	121.7	(5)	121.7	(5)
			c_H	4.6	4.7	(5)	4.7	(5)
		d_L	c_L	75.4	1696.3	(1)	5103.3	(5)
			c_H	19.6	144.2	(5)	144.2	(5)
	600	d_S	c_L	7.8	47.4	(5)	47.4	(5)
			c_H	2.6	2.3	(5)	2.3	(5)
		d_L	c_L	53.8	1467.3	(2)	2011.9	(5)
			c_H	13.8	47.0	(5)	47.0	(5)
Average (Sum)				23.7	441.4	(33)	935.3	(40)
1000	1000	d_S	c_L	24.5	887.5	(4)	933.4	(5)
			c_H	7.2	12.4	(5)	12.4	(5)
		d_L	c_L	150.0	1800.0	(0)	20483.1	(5)
			c_H	38.9	176.8	(5)	176.8	(5)
	1200	d_S	c_L	18.2	340.1	(5)	340.1	(5)
			c_H	5.5	8.3	(5)	8.3	(5)
		d_L	c_L	103.1	1800.0	(0)	5615.1	(5)
			c_H	28.1	126.6	(5)	126.6	(5)
Average (Sum)				46.9	644.0	(29)	3462.0	(40)
Total: Average (Sum)				35.3	542.7	(62)	2198.6	(80)

Table 9: Number of optimally solved instances and average computation times for Category (A2) for two different time limit settings: the classical $t_{\max} = 1800$ seconds vs. an open-end calculation with no time limit (indicated by $t_{\max} = \infty$).

We highlight the following main observations:

- The modeling times of these huge instances are still perfectly fine, as even the most challenging subset requires only 2.5 minutes on average. This is a justifiable effort in view of the expected solution time and gives hope that, in the future, possibly also these instances can be coped with more efficiently as a consequence of the steady progress in terms of optimization software.

- Still, 62 of the 80 instances are solved optimally within a maximum of 30 minutes. The unsuccessfully attempted instances, with only one exception, all originate from the constellation (d_L, c_L), which has already been identified as the greatest challenge before, with the help of Tab. 8. Comparing the two columns ($t_{\max} = 1800s$ vs. $t_{\max} = \infty$), it is noticeable that the only unsolved instance that did not come from that subset required a solution time that was roughly in the range of half an hour, so that one could also speak of a random effect here.

- Overall, it can be stated that on average all 80 instances are solved in less than one hour. The longest solution time we observed was slightly less than nine hours (for an instance with $n = 1000$ items and the combination (d_L, c_L)) – a time that is admittedly already relatively long, but nevertheless would not be achievable at all with the compact formulations from the literature.

Finally, we consider the instances from Category (B2) and first present an overview of the size of the corresponding CAF graphs in Tab. 10. On the one hand, it is noticeable that these networks are on average much smaller than was the case for the most difficult instances from Category (A2). By way of example, we mention that even for the most challenging combinations (e.g., Class X and $|T_S^{nd}| = 400$), one detects a much smaller size on average than, say, for $n = 1000$ items and the setting d_L in Category (A2). The instances from Category (B2) can thus still be classified as easier, despite a significant increase of the input parameter $|T_S^{nd}|$. It is also interesting to note that in contrast to Tab. 8, here a doubling of the input parameter $|T_S^{nd}|$ also leads to a doubling of the number of nodes and arcs for basically any instance class. This is mainly due to the fact that the number of items per clique for each class is predefined by the a-parameter choice (see [23] for the construction details) and thus, with increasing $|T_S^{nd}|$, only the number of cliques grows. Consequently, the graph size increases only proportionally to that value and not in a "combinatorial sense" (i.e., in the size of the cliques).

	number of nodes ($\|\mathcal{V}\|$)					number of arcs ($\|\mathcal{E}\|$)				
$\|T_S^{nd}\|$	200	250	300	350	400	200	250	300	350	400
I	6.5	8.4	10.0	11.8	13.6	8.0	10.3	12.2	14.5	16.7
II	19.6	24.1	27.6	31.7	35.8	26.5	32.4	37.2	42.7	48.2
III	34.3	44.6	55.6	62.8	71.7	44.8	58.7	73.0	82.5	94.4
IV	69.8	91.7	108.4	126.9	146.1	95.1	125.3	147.6	172.6	198.7
V	114.1	142.4	173.3	205.2	240.6	156.0	195.2	236.8	280.1	328.4
VI	83.8	107.1	129.6	151.3	171.2	179.2	229.7	277.5	324.9	367.4
VII	117.8	151.6	175.6	209.9	243.6	160.1	204.9	237.2	283.7	330.2
VIII	119.2	146.0	178.6	208.6	231.7	187.7	230.7	283.5	329.0	364.7
IX	82.4	104.4	123.6	145.3	165.2	184.0	234.0	275.5	322.5	366.8
X	194.4	248.3	299.2	343.1	386.6	301.6	386.1	467.3	535.6	602.1

Table 10: Average size of the CAF network for some instances from Category (B2) depending on the class index.

As a result of these observations, it seems reasonable to assume that even these enlarged instances will not be too challenging for CAF yet, and indeed this is also visible in the results depicted in Tab. 11. We highlight that in the case of TBPP, still every single instance can be solved optimally within a relatively short time (less than three minutes on average for any $|T_S^{nd}|$). Conversely, within the time limit of 1800 seconds, a very few instances of the TBPP-FU can no longer be dealt with because, as described earlier, solving this problem is generally somewhat more costly. We would like to note that most of the unsolved instances are from Class X and therefore, as seen previously in Tab. 10, correspond to the (on average) largest ILP models. In addition, especially for larger parameter settings (that is, approximately, $|T_S^{nd}| \geq 300$), sometimes one or two instances from Classes IV, V, or VII cannot be tackled successfully due to some random effects, so that there is no strict monotonicity in the number of instances solved to proven optimality for the TBPP-FU. However, and this is the difference to the instances from Category (A2), the maximum computation time in our case was only about 142 minutes (for one instance from Class X), so that we can assume that CAF will also solve the vast majority of even larger benchmark instances constructed according to the same principles.

Remark 11. *To get a somewhat more accurate idea of the actual boundaries of applying CAF to Category (B2), we conducted some further (less systematic) internal tests. In these calculations, we observed some first memory issues (in terms of storing the resulting branch-and-bound trees) when dealing with TBPP-FU instances having $|T_S^{nd}| = 500$ non-dominated starting times. However, this only happened in a very few exceptional cases, so that even here the size of the graph is not problematic, in general, and almost all instances can still be tackled properly.*

	TBPP		TBPP-FU	
$\lvert T_S^{nd}\rvert$	t	opt	t	opt
160	48.9	(100)	123.1	(100)
170	55.7	(100)	133.7	(100)
180	54.2	(100)	139.6	(100)
190	62.8	(100)	158.6	(100)
200	71.3	(100)	201.2	(99)
210	80.9	(100)	183.0	(99)
220	73.6	(100)	205.0	(99)
230	85.6	(100)	206.4	(99)
240	82.4	(100)	226.2	(99)
250	100.6	(100)	241.8	(98)
260	97.9	(100)	267.5	(98)
270	99.6	(100)	266.6	(97)
280	106.7	(100)	314.2	(96)
290	109.6	(100)	287.3	(96)
300	115.2	(100)	310.1	(96)
310	120.0	(100)	327.3	(96)
320	119.3	(100)	363.3	(94)
330	131.2	(100)	329.6	(97)
340	136.8	(100)	338.7	(98)
350	135.3	(100)	376.0	(95)
360	138.9	(100)	363.5	(96)
370	140.3	(100)	343.9	(98)
380	161.5	(100)	425.4	(94)
390	166.8	(100)	415.8	(95)
400	164.4	(100)	404.4	(97)
Average (Sum)	106.4	(2500)	278.1	(2436)

Table 11: Numerical Results for instances of Category (B2) for both, the TBPP and the TBPP-FU (with $t_{\max} = 1800s$).

6. Conclusions

In this article, we addressed the exact solution of two types of temporal bin packing problems, the TBPP and the TBPP-FU, by developing a new graph-theoretic approach (called CAF). Such an idea had previously been identified in the literature as an inefficient solution method given the generally exponential size of the resulting networks. By cleverly grouping equivalent states together in the construction of the graph, we managed to significantly reduce the number of nodes and arcs compared to previous concepts from [12] and [17]. Remarkably, the associated ILP formulations can now be generated in a relatively short time even for very large instances, and thus they can easily be passed to a commercial ILP solver. Based on extensive test calculations, it turns out that for the first time ever all benchmark instances of the TBPP and the TBPP-FU, previously used in the context of exact approaches, can be solved to proven optimality in reasonably short time. Moreover, our new formulation not only outperforms the previous state of the art in terms of solution times, but also succeeds in handling much larger new benchmark instances based on the classical test scenarios mentioned before. All in all, we have thus presented a powerful unified approach for solving temporal bin packing problems, the basic concepts of which can be prospectively applied (with minor modifications, if necessary) to other classes of optimization problems in the field of interval scheduling. In future research, we will try to further improve this very promising concept, for example by incorporating reduced cost variable fixing or by investigating whether the now known optimal solutions can also be obtained using thinned-out graphs, like for example illustrated for the CSP in [21, 24].

Declarations

Funding: This work has been supported by FCT – Fundação para a Ciência e Tecnologia within the R&D Units Project Scope UIDB/00319/2020.. **Conflicts of interest:** The authors declare that they do not have any conflicts of interest. **Availability of data and material:** For clarity, the instances used in this paper have been gathered together at `https://github.com/wotzlaff/tbpp-instances`. However, we note that most of these instances were originally designed in [3] and [23], and some of them were already available online, see `https://github.com/sibirbil/TemporalBinPacking`. **Code availability:** The instances were solved by the commercial software Gurobi. The underlying implementation of the models in Python can be found at `https://github.com/wotzlaff/tbpp-caf`.

References

[1] Angelelli, E., Bianchessi, N., Filippi, C.: Optimal interval scheduling with a resource constraint. Computers & Operations Research 51, 268–281 (2014)

[2] Arkin, E.M., Silverberg, E.B.: Scheduling jobs with fixed start and end times. Discrete Applied Mathematics 18(1), 1–8 (1987)

[3] Aydin, N., Muter, I., Ilker Birbil, S.: Multi-objective temporal bin packing problem: An application in cloud computing. Computers & Operations Research 121, Article 104959 (2020)

[4] Barnett Jr., T., Jain, S., Andra, U., Khurana, T.: Cisco Visual Networking Index (VNI) Complete Forecast Update, 2017–2022. APJC Cisco Knowledge Network (CKN) Presentation (2018) (`https://www.cisco.com/c/dam/m/en_us/network-intelligence/service-provider/digital-transformation/knowledge-network-webinars/pdfs/1213-business-services-ckn.pdf`)

[5] Barnhart, C., Johnson, E.L., Nemhauser, G.L., Savelsbergh, M.W.P., Vance, P.H.: Branch-and-Price: Column Generation for Solving Huge Integer Programs. Operations Research 46(3), 316–329 (1998)

[6] Bar-Noy, A., Bar-Yehuda, R., Freund, A., Naor, J.: A Unified Approach to Approximating Resource Allocation and Scheduling. Journal of the ACM 48(5), 1069–1090 (2001)

[7] Bar-Noy, A., Canetti, R., Kutten, S., Mansour, Y., Schieber, B.: Bandwidth Allocation with Preemption. SIAM Journal on Computing 28(5), 1806–1828 (1999)

[8] Bartlett, M., Frisch, A.M., Hamadi, Y., Miguel, I., Tarim, S., Unsworth, C.: The temporal knapsack problem and its solution. Lecture Notes in Computer Science 3524, 34–48 (2005)

[9] Biedl, T.: Graph-theoretic algorithms. Lecture notes, University of Waterloo (2005)

[10] Boschetti, M.A., Hadjiconstantinou, E., Mingozzi, A.: New upper bounds for the two-dimensional orthogonal non guillotine cutting stock problem. IMA Journal of Management Mathematics 13(2), 95–119 (2002)

[11] Buyya, R., Yeo, C.S., Venugopal, S., Broberg, J., Brandic, I.: Cloud computing and emerging IT platforms: vision, hype, and reality for delivering computing as the 5th utility. Future Generation Computer Systems 25(6), 599–616 (2009)

[12] Caprara, A., Furini, F., Malaguti, E.: Uncommon Dantzig-Wolfe reformulation for the temporal knapsack problem. INFORMS Journal on Computing 25(3), 560–571 (2013)

[13] Caprara, A., Furini, F., Malaguti, E., Traversi, E.: Solving the temporal knapsack problem via recursive Dantzig-Wolfe reformulation. Information Processing Letters, 116(5), 379–386 (2016)

[14] Chen, B., Hassin, R., Tzur, M.: Allocation of Bandwidth and Storage. IIE Transactions 24, 501–507 (2002)

[15] Clausen, J.V., Lusby, R., Ropke, S.: Consistency Cuts for Dantzig-Wolfe Reformulations. *to appear in:* Operations Research (2022) (`https://doi.org/10.1287/opre.2021.2160`)

[16] Clautiaux, F., Carlier, J., Moukrim, A.: New reduction procedures and lower bounds for the two-dimensional bin packing problem with fixed orientation. Computers & Operations Research 34(8), 2223–2250 (2007)

[17] Clautiaux, F., Detienne, B., Guillot, G.: An iterative dynamic programming approach for the temporal knapsack problem. European Journal of Operational Research 293(2), 442–456 (2021)

[18] Darmann, A., Pferschy, U., Schauer, J.: Resource allocation with time intervals. Theoretical Computer Science 411, 4217–4234 (2010)

[19] de Cauwer, M., Mehta, D., O'Sullivan, B.: The Temporal Bin Packing Problem: An Application to Workload Management in Data Centres. Proceedings of the 28th IEEE International Conference on Tools with Artificial Intelligence, 157–164, (2016)

[20] de Lima, V.L., Alves, C., Clautiaux, F., Iori, M., Valério de Carvalho, J.M.: Arc Flow Formulations Based on Dynamic Programming: Theoretical Foundations and Applications. European Journal of Operational Research 296(1), 3–21 (2022)

[21] de Lima V.L., Iori M., Miyazawa F.K.: Exact solution of network flow models with strong relaxations. *to appear in:* Mathematical Programming (2022) (`https://doi.org/10.1007/s10107-022-01785-9`)

[22] Dell'Amico, M., Delorme, M., Iori, M., Martello, S.: Mathematical models and decomposition methods for the multiple knapsack problem. European Journal of Operational Research 274(3), 886–899 (2019)

[23] Dell'Amico, M., Furini, F., Iori, M.: A Branch-and-Price Algorithm for the Temporal Bin Packing Problem. Computers & Operations Research 114, Article 104825 (2020)

[24] Delorme, M., Iori, M.: Enhanced pseudo-polynomial formulations for bin packing and cutting stock problems. INFORMS Journal on Computing 32(1), 101–119 (2020)

[25] Delorme, M. Iori, M., Martello, S.: Bin packing and Cutting Stock Problems: Mathematical Models and Exact Algorithms. European Journal of Operational Research 255, 1–20 (2016)

[26] Fettweis, G., Dörpinghaus, M., Castrillon, J., Kumar, A., Baier, C., Bock, K., Ellinger, F., Fery, A., Fitzek, F., Härtig, H., Jamshidi, K., Kissinger, T., Lehner, W., Mertig, M., Nagel, W., Nguyen, G.T., Plettemeier, D., Schröter, M., Strufe, T.: Architecture and advanced electronics pathways towards highly adaptive energy-efficient computing. Proceedings of the IEEE 107(1), 204–231 (2019)

[27] Furini, F.: Decomposition and reformulation of integer linear programming problems. PhD thesis, Università di Bologna (2011)

[28] Gilmore, P.C., Gomory, R.E.: A Linear programming approach to the cutting-stock problem (Part I). Operations Research 9, 849–859 (1961)

[29] Gschwind, T., Irnich, S.: Stabilized column generation for the temporal knapsack problem using dual-optimal inequalities. OR Spectrum 39, 541–556 (2017)

[30] Hall, N.G., Magazine, M.J.: Maximizing the value of a space mission. European Journal of Operational Research 78, 224–241 (1994)

[31] Kantorovich, L.V.: Mathematical methods of organising and planning production. Management Science 6, 366–422 (1939 Russian, 1960 English)

[32] Kaplan, J.M., Forrest, W., Kindler, N.: Revolutionizing data center energy efficiency. Technical report, McKinsey & Company (2008)

[33] Kellerer, H., Pferschy, U., Pisinger, D.: Knapsack Problems. Springer (2004)

[34] Kolen, A.W.J., Lenstra, J.K., Papadimitriou, C.H., Spieksma, F.C.R.: Interval scheduling: A survey. Naval Research Logistics 54(5), 530–543 (2007)

[35] Martinovic, J., Delorme, M., Iori, M., Scheithauer, G., Strasdat, N.: Improved flow-based formulations for the skiving stock problem. Computers & Operations Research 113, Article 104770 (2020)

[36] Martinovic, J., Strasdat, N., Selch, M.: Compact Integer Linear Programming Formulations for the Temporal Bin Packing Problem with Fire-Ups. Computers & Operations Research 132, Article 105288 (2021)

[37] Martinovic, J., Strasdat, N., Valério de Carvalho, J.M., Furini, F.: Variable and constraint reduction techniques for the temporal bin packing problem with fire-ups. to appear in: Optimization Letters (2021) (`https://link.springer.com/article/10.1007/s11590-021-01825-x`)

[38] Martinovic, J., Strasdat, N.: Theoretical Insights and a New Class of Valid Inequalities for the Temporal Bin Packing Problem with Fire-Ups. Preprint MATH-NM-01-2022, Technische Universität Dresden (2022) (`http://www.optimization-online.org/DB_HTML/2022/02/8791.html`)

[39] Phillips, C.A., Uma, R.N., Wein, J.: Offline admission control for general scheduling problems. Journal of Scheduling 3, 365–381 (2000)

[40] Ryan, D.M., Foster, B.A.: An integer programming approach to scheduling. Computer Scheduling of Public Transportation Urban Passenger Vehicle and Crew Scheduling, 269–280 (1981)

[41] Sadykov, R., Vanderbeck, F., Pessoa, A., Tahiri, I., Uchoa, E.: Primal heuristics for branch and price: The assets of diving methods. INFORMS Journal on Computing 31(2), 251–267 (2019)

[42] Scheithauer, G.: Introduction to Cutting and Packing Optimization – Problems, Modeling Approaches, Solution Methods. International Series in Operations Research & Management Science 263, Springer, 1.Edition (2018)

[43] Valério de Carvalho, J.M.: LP models for bin packing and cutting stock problems. European Journal of Operations Research 141(2), 253–273 (2002)

Appendix A. Further Illustrations

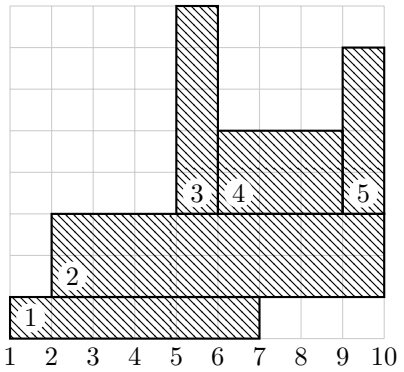

Figure A.10: Visualization of the instance E_2 from Example 5.

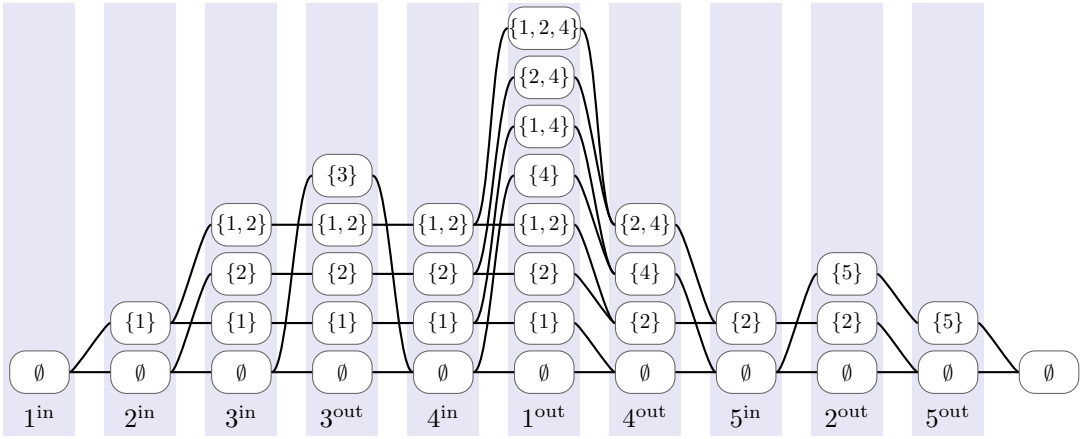

Figure A.11: The network from [17] when applied to instance E_2 from Example 5.

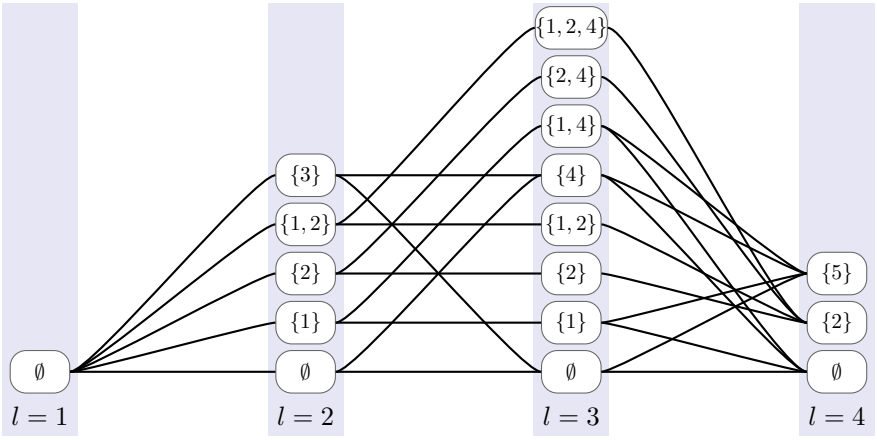

Figure A.12: The network from [12] when applied to the instance E_2 from Example 5.

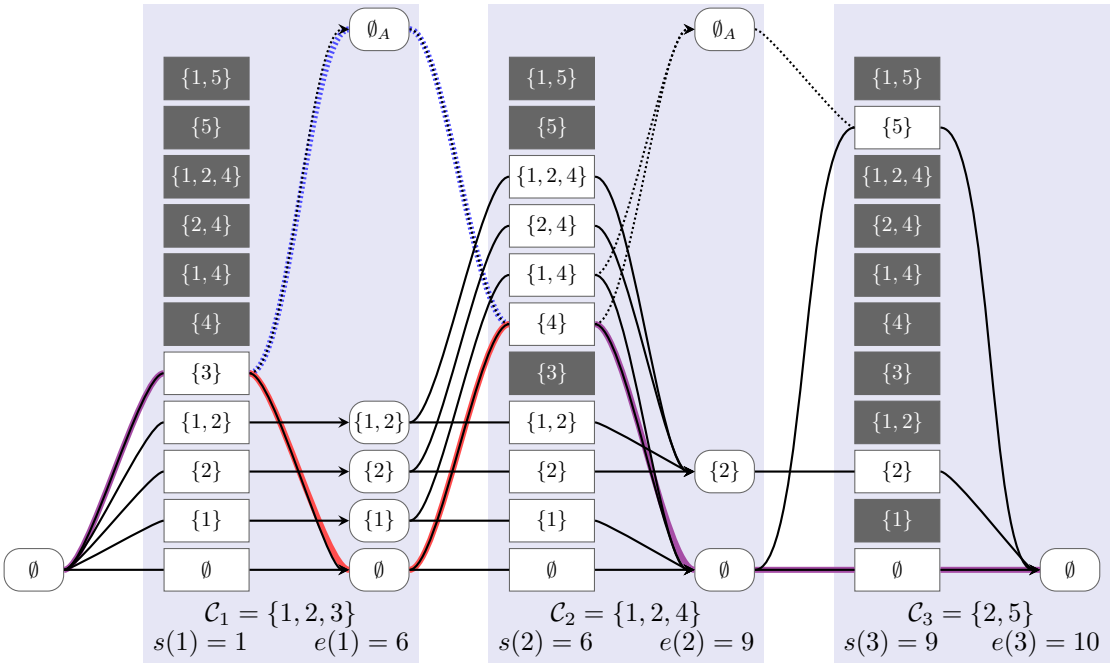

Figure A.13: The combinatorial arcflow graph for the TBPP-FU and the instance E_2 with two different paths leading to the same pattern $J = \{3, 4\}$. The joint (parts of the) arcs are colored purple, while the two alternatives to pass layer $l = 1$ are painted red and blue, respectively. In terms of optimization, the blue path will be preferred, because it does not imply additional fire-up costs (which is consistent with the pattern J described before).