Als Manuskript gedruckt

Technische Universität Dresden
Herausgeber: Der Rektor

Exact and Heuristic Solution Approaches for Busy Time Minimization in Temporal Bin Packing

John Martinovic, Nico Strasdat

Preprint MATH-NM-01-2023

September 2023

Exact and Heuristic Solution Approaches for Busy Time Minimization in Temporal Bin Packing

J. Martinovic[a,*], N. Strasdat[a]

[a]*Institute of Numerical Mathematics, Technische Universität Dresden, Germany*

Abstract

Given a set of jobs (or items), each of which being characterized by its resource demand and its lifespan, and a sufficiently large number of identical servers (or bins), the busy time minimization problem (BTMP) requires to find a feasible schedule (i.e., a jobs-to-servers assignment) having minimum overall power-on time. Although being linked to the field of temporal bin packing, BTMP represents an independent branch of research. Typically, such considerations (and generalizations of it) are very important in data center workload management to keep operational costs low. Hence, finding efficient and powerful solution techniques for BTMP is a relevant topic in cutting and packing, both from a theoretical and practical point of view. In this article, we give an overview of heuristic and exact approaches for the problem under consideration and analyze their theoretical properties and computational behavior. As a first main contribution, we suggest a new best-cost based heuristic showing convincing results in a wide variety of numerical test. In terms of exact approaches, we propose some improvements for the ILP models from the literature and establish a new combinatorial flow-based formulation. Based on extensive numerical tests with differently-characterized benchmark sets, the flow model is shown (i) to improve the state-of-the-art approach for general instances, and (ii) to be competitive with a matching formulation tailored for a special case.

Keywords: Cutting and Packing, Busy Time Minimization, Temporal Bin Packing, Flow Formulation, Scheduling

1. Introduction

We consider the *busy time minimization problem (BTMP)*: Given

- a set of $n \in \mathbb{N}$ *items (jobs)* having *workload (resource demand)* c_i and *lifespan (time window)* $[s_i, e_i)$, $i \in I := \{1, \ldots, n\}$, and
- a sufficiently large number of heterogeneous *bins (servers)* $k \in K$ with capacity C,

the task is to find a feasible jobs-to-servers assignment minimizing the total power-on time, hereinafter referred to as the *busy time*, of all servers in use. In a broader sense, it hence falls into the domain of temporal bin packing, see [2, 6], but is mathematically independent of the problem classes studied therein. The latter also applies in terms of practical applications. It is true that more or less all variants of temporal bin packing (including BTMP) can be motivated by the fact that the operational costs in data centers should be kept as low as possible, see [15, 25]. However, compared to the classical *temporal bin packing problem (TBPP)* which aims to minimize the number of servers in use, see [5], the scientific importance of BTMP originates

*Corresponding author

Email addresses: john.martinovic@tu-dresden.de (J. Martinovic), nico.strasdat@tu-dresden.de (N. Strasdat)

mostly from the field of optical networks. In this context, signals are to be transmitted in optical fibers, where different wavelengths have to be assigned to different users. To avoid interference and to meet other quality-of-service constraints, the number of signals transported simultaneously in a given optical fiber is generally limited, see [1, 17, 28]. To keep costs down, the length of the fiber optic cables required should be as short as possible.

Given the fact that the BTMP is an $\mathcal{NP}$-hard optimization problem, see [28], in addition to the already mentioned early publications there is a relatively large body of works dealing with heuristic approaches and corresponding theoretical properties. However, many of these approximation results known in the literature are only true if special cases of the overall problem are considered. Among others, the following restrictions have been studied in [11] and follow-up articles indicated in the list below:

- *proper instances*, meaning that the time window of any arbitrary but fixed job is never contained in the time window of another job, see [22]. In this case, the interval graph is called *proper*, see also [3].
- *clique instances*, meaning that there is a point in time when all given jobs are active, see [22, 25]. In this case, the interval graph is equivalent to a clique of size n.
- *bounded-length instances*, where the job durations $d_i := e_i - s_i$, $i \in I$, are bounded by some constant, see [1, 4, 17].

In this article, we will just deal with the case, where neither of these additional restrictions is explicitly required[1]. Nevertheless, a very important special case, which has to be considered in the context of this work, deals with so-called *unit-size instances.* Here, all given jobs request $c_i = 1$ resource unit, so that on the one hand such instances are very challenging, especially from a combinatorial point of view. On the other hand, the corresponding (feasible) solutions offer some more structural properties, so that such instances are also of theoretical interest and are used, among other things, for the construction of worst-case examples. From an overall point of view, a first-fit heuristic proposed in [11] is the most competitive one for "general" instances (not necessarily belonging to the unit-size case), whereas a non-polynomial greedy heuristic, introduced in [4], is tailored to the special case of unit-size items. To not overload this introductory section, the theoretical properties of both these heuristics will be discussed more thoroughly in the main part of this article (see Sect. 3). Although having bounded worst-case factors, these methods cannot be applied to online scenarios (which are also quite common in busy time minimization), nor do they align with the actual objective function (i.e., the overall busy time units) in the decision-making process. Therefore, as a first main contribution of this article, in Sect. 3 we propose a new *best-cost heuristic (BCH)* that addresses exactly these small deficiencies and, moreover, also leads to more convincing numerical results for various sets of benchmark instances.

Even though the considered problem is an important variant of temporal bin packing, there are extremely few works on exact solution approaches in the literature. In fact, only the very recent preprint article [23] deals with such approaches, but is ultimately still more concerned with determining lower bounds on the optimal value. However, the empirical quality of these bounds cannot be measured precisely for many instances, since the computation of optimal values is still based on a low-performance assignment model that has to be considered state-of-the-art. Only for the very special subcase of unit-size instances with half-capacity jobs, a rather powerful framework based on a matching formulation is given in [23]. From an overall point of view, a promising way (in cutting and packing) to define tractable ILP models is given by flow formulations. The theoretical advantages of such formulations were already recognized and

[1]This is because these further assumptions may typically lead to a much easier (sometimes even polynomially solvable) optimization problem, see [22, Lemma 3.1].

worked out in early publications, see [29], but their numerical verification could only take place with the advancing developments of powerful software tools and hardware components, see [27]. Nowadays, flow-based models are among the most extensively studied solution approaches for basic problems like the cutting stock problem [8, 9, 19], the skiving stock problem [16, 18], but also for a wide variety of other neighboring fields of research, see [7]. As for temporal bin packing, flow formulations have not been considered an efficient solution method for a long time, since the resulting number of nodes and arcs generally grow exponentially with the input data, see [12]. However, a novel state-space reduction method proposed in [21] can make this exponentiality controllable (to a larger extent) and thus leads to a strong ILP model for temporal packing problems for the first time. In particular, the resulting *combinatorial arcflow (CAF)* model presented therein is able to solve all previously known benchmark instances of the temporal bin packing problem (with and without fire-ups), the vast majority of them even in very short time. Hence, in Sect. 4, the applicability of this graph-theoretic idea to the field of busy time minimization will be studied thoroughly. In the numerical computations conducted in Sect. 5, we will show that CAF is able to solve many benchmark instances to proven optimality for the first time, thus representing a new state-of-the-art approach for general instances. Moreover, for the special case considered in [23], CAF turns out to be competitive with a tailored matching-based formulation. Given these outcomes, this work represents the first contribution systematically dealing with exact and heuristic solution approaches for busy time minimization.

2. Preliminaries

2.1. Theoretical Foundations

Let us consider a set $I = \{1, \dots, n\}$ of given items (jobs), each of which being specified by a time interval $[s_i, e_i)$ and an item size c_i, $i \in I$. These items have to be assigned to servers of capacity C, so that the overall power-on time of all servers in use is minimized. In temporal bin packing, the following generic sets are required.

- $T := \bigcup_{i \in I} \{s_i, e_i\}$, the set of all time instants,
- $T_S := \bigcup_{i \in I} \{s_i\}$, the set of all starting times,
- $I_t := \{i \in I \, : \, t \in [s_i, e_i)\}$, the set of all items active at time $t \in T$,
- $T_S^{nd} \subseteq T_S$, the set of all nondominated[2] starting times.

Remark 1. *In some publications, a so-called parallelism parameter $g \in \mathbb{N}$ is given, stating that at most g jobs are allowed to run simultaneously (i.e., at the same time) on any server, see [4, 11, 26]. Obviously, this parameter invokes an (implicit) capacity constraint which is given in a natural way in some job-to-server scheduling applications, see [25], but is also based in particular on the first description of this optimization problem in optical network design, see [1, 17, 28].*

In fact, most publications dealing with busy time minimization do not specify both parameters, g and C, explicitly. By way of example, for given C a natural choice of g can be obtained by setting

$$g = \max \left\{ \sum_{i \in I_t} x_i \, : \, \sum_{i \in I_t} c_i x_i \leq C, \, t \in T_S^{nd}, \, \boldsymbol{x} = (x_1, \dots, x_n)^\top \in \{0,1\}^n \right\}.$$

[2]We say that $t_2 \in T_S$ dominates $t_1 \in T_S$ with $t_1 < t_2$ whenever $I_{t_1} \subseteq I_{t_2}$ holds. This means that all items active at t_1 are still active at t_2.

By that, we ensure that the capacity itself already limits the degree of parallelism on a server. On the other hand, if only g is given, the capacity C is typically chosen so that any combination of at most g items can be executed at the same time, i.e., we would then have

$$C = \max \left\{ \sum_{i \in I_t} c_i x_i \,:\, \sum_{i \in I_t} x_i \leq g,\, t \in T_S^{nd},\, \boldsymbol{x} = (x_1, \ldots, x_n)^\top \in \{0,1\}^n \right\}.$$

So, whenever one of these parameters is not given explicitly in the context of this article, it can be calculated according to the rules mentioned before.

To represent a specific BTMP, we start with the following definition:

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

An important special case of the BTMP is given by the *unit-size case*, meaning that $c_i = 1$ holds for all $i \in I$. In that scenario, the roles of g and C actually coincide since cardinality and capacity units are counted identically. Unit-size instances play an important role in worst-case considerations of heuristic approaches, see [4, 11], and some of the benchmark sets to be introduced later in this section.

Let us proceed with some preparatory definitions to simplify the description of the overall problem.

Definition 2 ([4])**.** *The length of an interval $\mathcal{I} = [a, b)$ is $\operatorname{len}(\mathcal{I}) = b - a$. The span of $\mathcal{I}$ is given by $\operatorname{span}(\mathcal{I}) = b - a$.*

Obviously, for a single interval the meaning of its length and span coincide. However, the difference will become clear when generalizing these concepts to sets of intervals.

Definition 3 ([4])**.** *For a set $\mathcal{S} = \{\mathcal{I}_k\}_{k \in K}$ of intervals, its length is given by $\operatorname{len}(\mathcal{S}) = \sum_{k \in K} \operatorname{len}(\mathcal{I}_k)$. The span of $\mathcal{S}$ is defined as*

$$\operatorname{span}(\mathcal{S}) = \operatorname{len}\left(\bigcup_{k \in K} \mathcal{I}_k \right). \tag{1}$$

Remark 2. *Strictly speaking, the right-hand side in Equation (1) is not properly defined yet, because we only introduced the length of a single interval or the length of a set of intervals. In that formula, however, the union $\bigcup_{k \in K} \mathcal{I}_k$ of intervals can always be equivalently represented as a set $\mathcal{S}' = \{\mathcal{I}'_k\}_{k \in K'}$ of pairwise disjoint intervals. Thanks to this observation, no harm will arise from applying the length operator to unions of intervals.*

In a slight abuse of notation, we further write $\operatorname{len}(I')$ or $\operatorname{span}(I')$, whenever a subset $I' \subseteq I$ of jobs is concerned. This notation is justified, because we know that

$$\operatorname{len}(I') = \operatorname{len}(\mathcal{S}) = \sum_{i \in I'} (e_i - s_i), \qquad \operatorname{span}(I') = \operatorname{span}(\mathcal{S}) = \operatorname{len}\left(\bigcup_{i \in I'} [s_i, e_i) \right)$$

holds for $\mathcal{S} := \{[s_i, e_i)\}_{i \in I'}$.

Roughly speaking, the span of a set of jobs $I' \subseteq I$ is the "length" (or "measure") of the projection of these jobs (interpreted as rectangular shapes in the capacity-time-plane) onto the time axis. Since $\operatorname{len}(I')$ typically counts an active period of overlapping jobs multiple times, we always have $\operatorname{span}(I') \leq \operatorname{len}(I')$, and equality holds if and only if I' is made up of pairwise non-overlapping jobs.

As a consequence of these definitions, the BTMP can be formulated as follows in a set-theoretic manner: Find a partition of I into partition classes $\{I^{(k)}\}_{k \in K}$, so that

- not more than g jobs of $I^{(k)}$, $k \in K$, are active at any point in time,
- $\sum_{k\in K} \operatorname{span}(I^{(k)})$ is lowest possible among all feasible schedules.

Obviously, in this formulation the jobs of partition class $I^{(k)}$ correspond to the jobs assigned to bin $k \in K$. Note that we use a superscript k to avoid confusion with the sets I_t modeling the items active at time $t \in T$ defined before.

Let us now examine the BTMP more thoroughly from a mathematical point of view. To this end, remember that unit-size instances will play an important role throughout our theoretical sections. At first, we investigate the relationship between the BTMP and the traditional TBPP.

Theorem 3. *Let E be a unit-size instance of the BTMP, then the minimum number of bins required to schedule the jobs can be found in polynomial time.*

Proof. Note that an instance of the BTMP can also be described by its interval graph. In that interpretation, every job is represented by a node, and two nodes are connected by an edge if there is a temporal overlap between them. Then, the clique number $\omega \in \mathbb{N}$ of this interval graph is equivalent to the maximum number of jobs running in parallel. Hence, an optimal schedule will require precisely $\lceil \frac{\omega}{g} \rceil$ servers due to the unit-size case. The claim is now proved by remembering that the clique number of an interval graph can be found in polynomial time by a greedy-like node coloring algorithm, see [24]. $\square$

Contrary to this result, already in one of the very first application-oriented articles from optical network design, the following statement is given. However, the same result can also be found in more recent articles, specifically dealing with busy time minimization, see [26, Sect. 2.2].

Theorem 4 (see Theorem 2.3 in [28])**.** *Solving the BTMP is $\mathcal{NP}$-hard for any $g \geq 2$.*

Hence, unless $\mathcal{P} = \mathcal{NP}$, there have to be instances for which any solution of BTMP uses more servers than required in TBPP, meaning that the two problems are in fact different. One such instance is given in the following example.

Example 1. *Let us consider the unit-size instance E with $g = 2$ depicted in Fig. 1.*

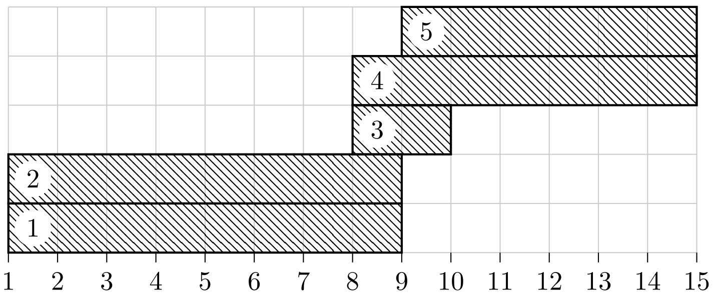

Figure 1: An exemplary instance for which BTMP uses more server than required by TBPP.

As regards the TBPP, two servers are sufficient to schedule all jobs feasibly. Indeed, a corresponding solution is given by $I^{(1)} = \{1, 2, 5\}$ on server $k = 1$ and $I^{(2)} = \{3, 4\}$ on server $k = 2$. To develop a better understanding of the notation introduced so far, by way of example, we note that $\operatorname{len}(I^{(1)}) = 8 + 8 + 6 = 22$, *whereas* $\operatorname{span}(I^{(1)}) = 8 + 6 = 14$. *Similarly, we have* $\operatorname{len}(I^{(2)}) = 2 + 7 = 9$ *and* $\operatorname{span}(I^{(2)}) = 7$. *Hence, the total busy time of the above TBPP solution is given by* 21 *units. On the other hand, there is exactly one optimal solution for the BTMP, and it is constructed by using three servers with $\widetilde{I}^{(1)} = \{1, 2\}$, $\widetilde{I}^{(2)} = \{4, 5\}$, and $\widetilde{I}^{(3)} = \{3\}$. The total busy time of this configuration results to* $\operatorname{span}(\widetilde{I}^{(1)}) + \operatorname{span}(\widetilde{I}^{(2)}) + \operatorname{span}(\widetilde{I}^{(3)}) = 8+7+2 = 17 < 21$.

Remark 5. *To also separate the problem under consideration from temporal bin packing with fire-ups, see [2], it is sufficient to note that (in BTMP), we can assume that an optimal solution does not contain any server that is activated more than once. An important corollary of this result is given by the fact that an instance of the BTMP can be decomposed (in a temporal sense), whenever* $\bigcup_{i \in I}[s_i, e_i)$ *does not form a single interval which is typically impossible when fire-ups have to be considered, see [20].*

Having established the BTMP as an independent branch of research, we conclude this subsection by some basic observations that will later be applied in the construction of exact approaches.

Definition 4 ([22]). *Let* $\mathcal{S}$ *be a set of intervals, then we define the savings of* $\mathcal{S}$ *by* $\operatorname{sav}(\mathcal{S}) = \operatorname{len}(\mathcal{S}) - \operatorname{span}(\mathcal{S})$.

The concept of savings will typically be applied to a set $I^{(k)}$ of jobs scheduled to some server $k \in K$. Then, the savings[3] $\operatorname{sav}(I^{(k)})$ somewhat measure the amount of overlapping among the jobs contained in $I^{(k)}$. Note that the savings are always nonnegative, because $\operatorname{span}(I^{(k)}) \leq \operatorname{len}(I^{(k)})$ holds for every server. With this new concept, we can derive a kind of dual perspective on the optimization problem considered in this chapter.

Theorem 6. *Let* E *be an instance of the BTMP. Then, minimizing the total busy time is equivalent to maximizing the total savings.*

Proof. Let us consider a feasible jobs-to-server assignment $\mathcal{S} := \{I^{(k)}\}_{k \in K}$, with $I^{(k)} \subseteq I$ for any $k \in K$, belonging to E. Then, the total savings of that configuration can be calculated as follows:

$$\begin{aligned} \operatorname{sav}(\mathcal{S}) &= \sum_{k \in K} \operatorname{sav}(I^{(k)}) = \sum_{k \in K} \left(\operatorname{len}(I^{(k)}) - \operatorname{span}(I^{(k)}) \right) = \sum_{k \in K} \operatorname{len}(I^{(k)}) - \sum_{k \in K} \operatorname{span}(I^{(k)}) \\ &= \operatorname{len}(I) - \sum_{k \in K} \operatorname{span}(I^{(k)}) = \operatorname{len}(I) - \operatorname{span}(\mathcal{S}). \end{aligned}$$

Hence, the total savings $\operatorname{sav}(\mathcal{S})$ are maximal if and only if the total busy time $\operatorname{span}(\mathcal{S})$ is as small as possible. □

Example 2. *Let us come back to the instance given in Fig. 1 and its optimal solution* $\widetilde{I}^{(1)} = \{1, 2\}$, $\widetilde{I}^{(2)} = \{4, 5\}$, *and* $\widetilde{I}^{(3)} = \{3\}$ *constructed in the previous example. The savings on any server are given by* $\operatorname{sav}(\widetilde{I}^{(1)}) = 8$, $\operatorname{sav}(\widetilde{I}^{(2)}) = 6$, *and* $\operatorname{sav}(\widetilde{I}^{(3)}) = 0$. *It can be shown that this is indeed the maximum amount of overlapping in a feasible schedule.*

2.2. Benchmark Sets

Since the BTMP is a relatively young field of research, there is not yet a comprehensive collection of associated benchmark instances. However, in the relevant literature, three major data sets have been introduced to study the behavior of exact approaches of different variants of temporal bin packing problems, see [21, 23]. Even if not all of them were specifically designed for busy time minimization, here we will consider the following two instance categories:

(A) In [2, Sect. 5], the authors presented 240 instances with $C = 100$, not belonging to the unit-size case. Although a finer categorization would be possible[4], for the purposes of this paper it is sufficient to divide these instances into 12 groups of 20 instances each.

[3]Given the fact, that the length and the span operator have been extended to subsets of jobs before, terms like $\operatorname{sav}(I')$ for $I' \subseteq I$ are well-defined, too.

[4]In the original source, Category (A) consists of 48 groups of 5 instances each. These instances are classified on the basis of four parameters. However, our internal tests (not reported here) have shown that the numerical impact of two of these parameters is not very pronounced. Hence, to also make the resulting tables a little clearer, we decided to only focus on the two main features here.

Any such group is defined by a pair (n, c_i), where the number of items is chosen from $n \in \{50, 100, 150, 200, 500, 1000\}$ and c_i is replaced by a label c_L or c_H referring to a *low* (i.e., $c_i \in [25, 50] \cap \mathbb{Z}_+$) or a *high* (i.e., $c_i \in [25, 75] \cap \mathbb{Z}_+$) resource demand of the items.

(C) In [23], a set of 150 unit-size instances with $g = C = 2$ is introduced for busy time minimization. The whole set is parametrized by two parameters (n, λ), with n denoting the number of items, and λ representing the expected lifetime (that is, the duration $d_i := e_i - s_i$) of any job. To be more precise, the lifetime of any fixed item is drawn from a geometric distribution with success probability $1/\lambda > 0$. Moreover, the starting time of any job is chosen from $[0, n] \cap \mathbb{Z}_+$ using a uniform distribution, and the item size is always set to $c_i = 1$, $i \in I$. According to these rules, note that five instances are generated for any fixed pair (n, λ), and the specific values of these parameters can be found in the following list:

- (C1) A first subset of instances consists of the parameter constellations (n, λ) with $n \in \{200, 400, 600\}$ and $\lambda \in \{5, 10, 20, 40, 80\}$. Thus, a total number of 75 instances is contained in this subclass. These instances have been attempted in the context of exact solution approaches in [23].
- (C2) A second subset of instances consists of the parameter constellations (n, λ) with $n \in \{800, 1200, 1600, 2000, 2400\}$ and $\lambda \in \{5, 10, 20\}$. A subset of these 75 instances has been studied numerically in [23], but only heuristically.

Remark 7. *In the literature, there is a further benchmark set for the classical TBPP, introduced in [6] and referred to as Category (B) in [21]. This set also consists of instances not belonging to the unit-size case and, therefore, would not help to gain further insights that are substantially different from those related to Category (A).*

3. Heuristic Approaches

3.1. An Overview of Heuristics from the Literature

The first heuristic studied for the general case was described in [14] and consists of dividing the set of jobs into two classes, *wide jobs* and *narrow jobs*. The terms wide and narrow refer to the item sizes c_i, and the classification is done by some appropriately chosen parameter $\alpha \in [0, 1]$ that is multiplied with the bin capacity C. Hence, a job $i \in I$ is called wide, if $c_i > \alpha \cdot C$ and narrow otherwise. In the algorithm presented in [14], the wide and narrow jobs are assigned to separate sets of servers. More precisely, the wide jobs are scheduled arbitrarily to their set of servers, whereas the narrow jobs are positioned on a disjoint set of servers according to a first-fit (FF) algorithm. For $\alpha = 0.25$, it was shown that this procedure leads to a 5-approximation of the actual optimal value, see [14, Theorem 9]. For two reasons, we will not present a more detailed consideration of this heuristic approach: On the one hand, it is not tailored to cope with the unit-size case, because all jobs appear in precisely one of the two classes, ending up with either a “random” assignment (in case all jobs are wide) or a pure FF algorithm (in case all jobs are narrow). However, and this is the (more important) second reason, the first-fit case was studied more successfully in the same year by the authors of [11]. To be more precise, the idea presented in Alg. 1 was introduced in that publication. Since the items in Alg. 1 are sorted with respect to non-increasing lengths (job durations), the corresponding heuristic will be referred to as FF-L.

Algorithm 1 FF-L for Busy Time Minimization

```
Input: Instance $E$ with item list ordered by non-increasing job durations $d_i$, where ties are
    broken by the lower item index.
1: Initialize an empty bin $k = 1$.
2: for $i \in I$ do
3:     Assign the currently considered item to the lowest-indexed bin that is able to accom-
       modate it. If no such bin exists, open a new bin and put item $i$ into it.
4: end for
Output: heuristic solution with objective value $FF\text{-}L(E)$.
```

Despite representing a relatively easy heuristic approach, the precise approximation factor of Alg. 1 is still an open question in cutting and packing.

Theorem 8 (see Theorem 2.1 and Theorem 2.4 in [11])**.** *The worst-case performance ratio of Alg. 1 is at least* 3 *and at most* 4.

Remark 9. *In [26, Theorem 3], a slight variation of Alg. 1 is claimed to possess a worst-case performance ratio of* 3*, which seems to close the open question raised above. However, the inequality chain used in this proof is not justified. To be more precise, the main idea of that proof is given by observing*

$$\operatorname{span}(I^{(1)}) \leq \frac{3}{g} \cdot \operatorname{len}(I^{(k')}), \tag{2}$$

where $k' \in K$ is the highest-indexed server that received an item by the FF-L algorithm. In fact, this statement is not correct, and neither is the corresponding proof. Indeed, the FF-L algorithm applied to the instance from Fig. 1 is sufficient to obtain a contradiction to Inequality (2).

For instances belonging to the unit-size case, another type of heuristics has been established in the literature, see [4]. To this end, let us define the following.

Definition 5 ([4])**.** *A track $\mathcal{T}$ is a set of jobs with pairwise non-overlapping time windows.*

Given a feasible solution of the BTMP with unit-size items, one can think of each server as the union of (at most) g individual tracks of jobs. Hence, the main idea of Alg. 2 from [4] is given by iteratively finding such tracks and grouping g of them together on some server. More precisely, this heuristic repeatedly identifies tracks of maximum length (among the items still available), and assigns g successive tracks to the same bin. By that, the hope is that bundling roughly equally long tracks may produce large overlaps (i.e., large savings and small busy times).

Algorithm 2 Greedy Track Heuristic (GTH) for Busy Time Minimization

```
Input: Instance $E$.
1: Open a first empty bin.
2: while $I \neq \emptyset$ do
3:     Compute the longest track $\mathcal{T}$ in $I$ and assign it to the current bin. Update set $I$ by
       removing the items contained in $\mathcal{T}$.
4:     If the current bin contains exactly $g$ tracks after that, close it and open a new one.
5: end while
Output: heuristic solution with objective value $GTH(E)$.
```

Note that a discrete optimization problem has to be solved in any iteration of Alg. 2. Hence, this heuristic cannot be performed in polynomial time. However, these additional efforts may possibly lead to a better worst-case behavior, at least for a special case.

Theorem 10 (see Theorem 5 in [4])**.** *Let E be a unit-size instance of the BTMP, then we have $GTH(E) \leq 3 \cdot OPT(E)$.*

In [4, Subsect. 4.3], the tightness of the factor 3 is shown, but only for a generalized setting in which the given unit-size jobs possess an additional processing time, so that they can be assigned arbitrarily within their time interval as long as the processing time is respected. This setting is referred to as *busy time minimization with flexible jobs.* However, it is not known whether a similar result also holds for the scenario we consider.

Altogether, to the best of our knowledge, the 3-approximation given in the previous result is the currently best worst-case performance ratio known for unit-size instances of the BTMP, if neither of the additional restrictions discussed in Sect. 1 is part of the assumptions. In particular, the proofs given in the appendix of [4] suggest that the 2-approximations derived in that article at least make use of bounded interval lengths, since the concepts are directly inherited from [1, 17].

3.2. A Best-Cost Heuristic

The heuristic approaches investigated so far have primarily used the length of the jobs as a central decision criterion. The intention behind these approaches is quite plausible in the context of BTMP; after all, particularly the long items may contribute many busy time units to the objective function and should therefore be placed cleverly. Moreover, we have seen that the heuristics deduced from that logic exhibited bounded worst-case behavior.

Nevertheless, not only the length of the items is important, but also their concrete positioning within the considered time horizon, for example measured by the starting time s_i. In fact, jobs with similar starting times are likely to share a common time window, so that savings in busy time can be expected by grouping such jobs together. Thus, we would like to contrast the heuristics studied so far with one that focuses in particular on the temporal relation between items. To this end, we consider the *best-cost heuristic (BCH)* introduced in Alg. 3. We will refer to that specific algorithm by the abbreviation BCH-T, because the sorting is done with respect to the starting times s_i.

Algorithm 3 BCH-T for Busy Time Minimization

```
Input: Instance $E$ with item list ordered by non-decreasing starting times $s_i$, where ties are
    broken by the lowest item index.
1: Open a first empty bin.
2: for $i \in I$ do
3:     Among the bins that can accommodate item $i$, find that one in which item $i$ currently
       adds the lowest costs to the objective function (breaking ties by the lowest bin index).
       If there is no such bin, then open a new one and place item $i$ therein.
4: end for
Output: heuristic solution with objective value $BCH\text{-}T(E)$.
```

In it, we place the current item on a "locally optimal" server, meaning that adding this item to the intended bin produces the smallest possible increase in terms of total busy time units. If there are several such servers, we choose the lowest-indexed one.

From a theoretical point of view, this heuristic can be shown to possess an unbounded worst-case behavior.

Theorem 11. *The best-cost heuristic from Alg. 3 has an unbounded worst-case performance ratio.*

Proof. Let us consider a unit-size instance $E(\alpha, g)$, parametrized by the parallelism parameter $g \geq 2$ and some sufficiently large $\alpha \geq g^2$. The construction details are as follows:

- $E(\alpha, g)$ consists of g blocks of g items each. Any block possesses one long item of length α and $g - 1$ short items of maximum length $2g - 1$.

- More precisely, the items of block $p \in \{1, \ldots, g\}$ are given by one item with $[s_i, e_i) = [2p-1, 2p-1+\alpha)$ (that is, the long item) and $g-1$ items with $[s_i, e_i) = [2p, 2g+1)$ (the short items).

Altogether, an instance $E(\alpha, g)$ covers exactly $2g-2+\alpha$ units of time, and it always has a total number of g^2 items which all share the common time window $[2g, 2g+1)$. In other terms, any instance $E(\alpha, g)$ is a clique instance since there is only one maximal clique collecting all the items. For the sake of exposition, one specific instance from that family is illustrated in Fig. 2.

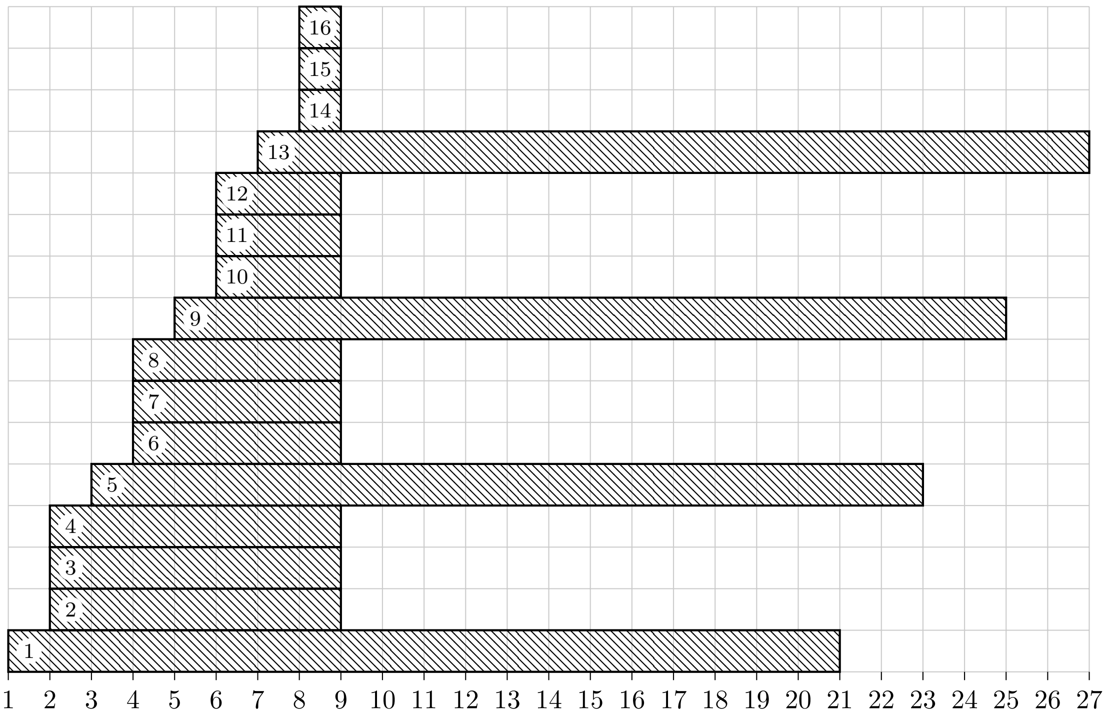

Figure 2: An illustration of $E(\alpha, g)$ for $g = 4$ and $\alpha = 20 > g^2$. The instance consists of $g = 4$ structurally more or less identical blocks, and the items are numbered with respect to non-decreasing starting times. Note that the time horizon has a total length of $2g - 2 + \alpha = 6 + 20 = 26$ units.

Now let us compute the heuristic value and the corresponding optimal busy time:

- Since all the items share a common time interval, in any iteration of Alg. 3, there is precisely one possibility to place the current item. As a consequence of that, BCH leads to a packing, where any block of g successive items is placed in a separate bin. This leads to
$$BCH\text{-}T(E(\alpha, g)) = g \cdot \alpha,$$
because there are g such blocks and the busy time of any server is determined by the single long item assigned to it.

- In an optimal solution, all long items will be assigned to the same server (for simplicity labelled $k = 0$), leading to a busy time of
$$(2g - 1 + \alpha) - 1 = 2g - 2 + \alpha,$$
because the entire time horizon is covered. The remaining $g \cdot (g-1)$ items fit on $g-1$ servers, and an optimal assignment is given by:

– Server $k = 1$ of that family has $g - 1$ items of block $p = 1$ and one item of block $p = 2$. This leads to $2g + 1 - 2 = 2g - 1$ busy time units.

– Server $k = 2$ of that family has $g - 2$ items of block $p = 2$ and two items of block $p = 3$. This leads to $2g + 1 - 4 = 2g - 3$ busy time units.

– In general, server k of that family has $g - k$ items of block $p = k$ and k items of block $p = k + 1$. This leads to $2g + 1 - 2k$ busy time units.

The total busy time is then given by

$$\begin{aligned} OPT(E(\alpha, g)) &= (2g - 2 + \alpha) + \sum_{p=1}^{g-1} (2g + 1 - 2p) \\ &= (2g - 2 + \alpha) + (g - 1)(2g + 1) - 2 \cdot \sum_{p=1}^{g-1} p \\ &= (2g - 2 + \alpha) + (g - 1)(2g + 1) - g(g - 1) \\ &= (2g - 2 + \alpha) + (g - 1)(g + 1) = (g - 1)(g + 3) + \alpha. \end{aligned}$$

Hence, we obtain

$$\frac{BCH\text{-}T(E(\alpha, g))}{OPT(E(\alpha, g))} = \frac{g \cdot \alpha}{(g - 1)(g + 3) + \alpha}$$

which equals $\mathcal{O}(g)$ (due to $\alpha \geq g^2$). Thus, the approximation factor can be arbitrarily bad. □

Remark 12. *Note that the previous result is independent of the tie-break rule since all the items having the same starting time are identical.*

Despite this supposed (theoretical) disadvantage, Alg. 3 has two major advantages compared to the heuristics from the literature. At first, it does not require full information about all incoming jobs, meaning that it can also serve as an online heuristic. Secondly, it directly addresses the actual objective function of BTMP when making the iteration-based decisions. Especially the latter aspect turns out to be beneficial in numerical tests.

3.3. Computational Results

In the following practical investigation of heuristics, we restrict ourselves to those approaches having polynomial running time. This is mainly due to the fact that we will also consider instances that do not belong to the unit-size case (that is, Category (A)), and the latter instances cannot be processed by the non-polynomial greedy track heuristic from Alg. 2. Overall, the heuristics to be empirically investigated are the following:

- BCH-T: Best Cost Heuristic with item list sorted by non-decreasing starting times (see Alg. 3).
- BCH-L: Best Cost Heuristic with item list sorted by non-increasing item lengths (instead of non-decreasing starting times)
- FF-T: First Fit Heuristic with item list sorted by non-decreasing starting times (instead of non-increasing item lengths)
- FF-L: First Fit Heuristic with item list sorted by non-increasing item lengths (see Alg. 1)

Overall, any of these heuristics is specified by a label consisting of the "type" of heuristic and an indicator pointing out either that the length of the jobs ('L') or their starting time ('T') was used as the sorting criterion. Through this experimental setup, we not only compare the different philosophies of the heuristics themselves, but also examine how they respond to a change in the order of the inputs. The empirical performance of the heuristic procedure itself can thus be better separated from the influence of the pre-sorting.

For the sake of exposition, let us start with the instances from Category (C2) tailored for the BTMP. Remember that $g = C = 2$ holds, and they all belong to the unit-size case. The relative objective values obtained by the four heuristics considered are gathered in Tab. 1.

n	λ	BCH-T	BCH-L	FF-T	FF-L
800	5	**0.0%**	3.3%	6.9%	4.0%
	10	**0.0%**	2.7%	6.4%	2.8%
	20	**0.0%**	3.6%	5.9%	4.1%
Average		**0.0%**	3.2%	6.4%	3.6%
1200	5	**0.0%**	3.2%	6.2%	3.8%
	10	**0.0%**	3.4%	7.3%	4.0%
	20	**0.0%**	3.5%	5.5%	4.1%
Average		**0.0%**	3.4%	6.3%	4.0%
1600	5	**0.0%**	2.9%	6.9%	3.6%
	10	**0.0%**	3.5%	7.1%	4.3%
	20	**0.0%**	4.0%	6.1%	4.5%
Average		**0.0%**	3.5%	6.7%	4.1%
2000	5	**0.0%**	2.8%	6.8%	3.5%
	10	**0.0%**	3.2%	7.0%	3.8%
	20	**0.0%**	3.9%	6.6%	4.3%
Average		**0.0%**	3.3%	6.8%	3.9%
2400	5	**0.0%**	2.9%	6.9%	3.7%
	10	**0.0%**	3.3%	6.9%	3.9%
	20	**0.0%**	3.7%	6.1%	4.1%
Average		**0.0%**	3.3%	6.6%	3.9%

Table 1: Average percentage deviation from the best heuristic value (among the four heuristics studied) for Category (C2)

Remark 13. *Note that we do not report the computation times required to execute the various heuristics, since the maximum time taken (by any of the four heuristics) to solve an instance of Category (C2) was slightly larger than one second. Moreover, the only noticeable difference in the running times was that any first-fit variant was marginally faster than its best-cost counterpart. This is due to the fact that only one possible bins to pack the current item has to be identified.*

In terms of the objective values, the main observations can be summarized as follows:

- Although there are only moderate differences between the various heuristic values in Tab. 1, we can deduce that BCH-T was able to find the best feasible solution for every single instance of (C2). However, there is no strict dominance, because there are a few instances from (C1), not tabulated here, where BCH-T did not offer the best heuristic result.
- Overall, the best-cost heuristics are typically better than their first-fit partners. This behavior is plausible since the iterative decisions in first-fit are not directly linked to the actual objective function of the BTMP. By that, we mean that heuristics of first-fit type just make sure to use a small number of bins in total, but they do not specifically focus on small busy times. The latter aspect is only part of the best-cost heuristics.

- Interestingly, the performances of BCH-L and FF-L are rather similar (with slight advantages for the best-cost version), whereas BCH-T and FF-T lead to the best and worst heuristic value, respectively. Hence, for the instances of Category (C2), the performance gain from changing the heuristic approach (from FF to BCH) heavily depends on the chosen pre-sorting. The main reason for this behavior is due to the relatively large item sizes ($C = 2c_i$ for all $i \in I$). Hence, if jobs are sorted according to criterion 'L', first-fit and best-cost will do the same for quite some time, especially because many jobs will fit on the first server (given the different time windows occupied by the arriving items). However, as long as just one server is open, BCH and FF cannot lead to different assignments. For setting 'T', the probability of sharing some joint interval on the time axis is higher for consecutive jobs. Hence, it is more likely that already the first few jobs will lead to more than one open server. From then on, BCH typically assigns the items more cleverly so that the performance gain is much more pronounced.
- The numbers presented suggest that the parameter λ does not have any influence on the performance of the heuristics. However, another experiment (not reported here in details) with Category (C1) offering more choices for λ showed that, on average, the differences between the heuristic values become smaller for increasing λ. In these cases, the job durations are typically longer, and so (much) more feasible combinations of items will lead to considerable temporal overlaps, meaning that there are (much) more rather dense packings having good objective values.

Let us now check whether a similar behavior can be observed for the instances from Category (A). We emphasize that these instances do not belong to the unit-size case. To be more precise, a maximum of four jobs can fit on one server at the same time, given the description of the jobs with $c_i \geq 25$ and $C = 100$. As regards the running times, we just mention that all instances are solved in less than half a second[5]. The relative quality of the feasible points obtained by the various heuristics can be evaluated by the numerical data collected in Tab. 2.

c_i	n	BCH-T	BCH-L	FF-T	FF-L
c_L	50	**0.0%**	8.5%	4.0%	9.1%
	100	**0.0%**	10.2%	3.5%	10.9%
	150	**0.0%**	11.0%	3.7%	11.0%
	200	**0.0%**	12.0%	4.3%	12.0%
	500	**0.0%**	12.2%	4.3%	12.3%
	1000	**0.0%**	12.4%	4.2%	12.7%
Average		**0.0%**	11.0%	4.0%	11.3%
c_H	50	**0.0%**	3.5%	5.7%	7.9%
	100	**0.0%**	4.1%	6.1%	7.5%
	150	**0.0%**	4.4%	7.3%	7.6%
	200	**0.0%**	5.5%	7.0%	8.2%
	500	**0.0%**	5.6%	7.8%	9.0%
	1000	**0.0%**	5.9%	8.2%	9.4%
Average		**0.0%**	4.8%	7.0%	8.3%
Total: Average		**0.0%**	7.9%	5.5%	9.8%

Table 2: Average percentage deviation from the best heuristic value (among the four heuristics studied) for Category (A). The averages are based on 20 instances each.

[5]The time is smaller than for Category (C2) since the number of items is not as large in Category (A).

Here, the main observations are given as follows:

- At first, we again see that BCH-T leads to the best objective value for any considered instance. Compared to the results of Tab. 1, the "lead" of BCH-T is now even more pronounced compared to its competitors. By that, we mean that the average percentage gaps of at least two of the other heuristics have become much larger. Only for FF-T, the percentage gap has not changed significantly.
- For low-capacity jobs ('c_L'), we see that there is almost no difference between BCH-L and FF-L anymore. In general, the reason for that is similar to the explanations for Category (C2), but here up to four jobs can be packed in parallel. Hence, BCH-L and FF-L will perform precisely the same steps for much more iterations, until a new server has to be opened. From then on, the algorithms can lead to different assignments.
- Interestingly, the FF-T algorithm seems to benefit from low-capacity items ('c_L'). We attribute this to the fact that, in this scenario, many consecutive jobs can be packed together on the same server so that dense packings (in the capacity-dimension) can be obtained rather easily. For instances having large item sizes ('c_H'), the situation is comparable to the (C2) instances, where FF-T was not so convincing.
- In a simplified way, one could state that the sorting criterion 'T' is important for small items, whereas the best-cost paradigm is important for large items. Hence, the heuristic combining these two features (BCH-T) performs best on average for Category (A), too.

As a consequence of our experiments, BCH-T (from Alg. 3) seems the most adequate heuristic to handle the benchmark sets considered here. Thanks to these observations, we will apply that approach to provide a feasible starting point for the exact solutions examined in the next section.

4. Exact Approaches

Very recently, the first systematic approach to introduce exact formulations for the BTMP was presented in [23]. Given the fact that the problem considered is already $\mathcal{NP}$-hard for $g = 2$, the authors of that article just considered unit-size instances with $g = C = 2$. However, the compact models presented therein already fail to solve moderately-sized instances, so that the article focuses more on lower bounds for the problem under consideration. In this section, we start by repeating an ILP formulation introduced in [23], and then move forward to transferring the concept of CAF networks from [21] to this new optimization problem. Given the focus of the aforementioned research article, a separate subsection will also be devoted to the special case $g = 2$. Interestingly, in that specific scenario, our network-based approach can be equivalently formulated as a *savings formulation*, the foundations of which were already partly contained in [23]. Overall, we will see that, on the one hand, CAF is more general in the sense that it is not tailored for $g = 2$, but, on the other hand, it is also competitive with the currently best formulation available for the aforementioned special case.

4.1. An ILP Model from the Literature

A classic way to deal with assignment problems is based on the idea of Kantorovich proposed in the context of one-dimensional bin packing, see [13]. Clearly, a similar strategy can be used to formulate a first compact ILP model for the BTMP. To this end, let $x_{ik} \in \{0, 1\}$ denote whether item $i \in I$ is assigned to bin $k \in K$. To accurately keep track of the busy time of the servers, we let $\Theta = \{1, \ldots, |T|\}$ represent an index set of T. Without loss of generality, we assume the elements of T to be ordered chronologically, meaning that $T = \{t_1 < t_2 < \ldots < t_{|T|}\}$ holds. Note that the activity status of a server can only change at precisely those instants of time. In other words, a server always stays busy (or empty) during the entire interval $[t_\theta, t_{\theta+1})$ for $\theta \in \Theta \setminus \{|T|\}$. Hence, we introduce the set of decision variables $y_{\theta,k} \in \{0, 1\}$ with $y_{\theta,k} = 1$ if and only if server $k \in K$ is active during the interval $[t_\theta, t_{\theta+1})$, $\theta \neq |T|$. Note that it is not sufficient to deal with non-dominated starting times in this setting, because we have to correctly count any instant of time with a positive load on a server.

According to [23], with these definitions we obtain the

Assignment Model for the BTMP

$$z^{ass} = \sum_{k \in K} \sum_{\theta \in \Theta \setminus \{|T|\}} (t_{\theta+1} - t_\theta) \cdot y_{\theta,k} \rightarrow \min$$

$$\begin{aligned}
\text{s.t.} \quad & \sum_{k \in K} x_{ik} = 1, && i \in I, && (3)\\
& \sum_{i \in I_{t_\theta}} x_{ik} \leq g \cdot y_{\theta,k}, && \theta \in \Theta \setminus \{|T|\},\, k \in K, && (4)\\
& x_{ik} \in \{0,1\}, && i \in I,\, k \in K, && (5)\\
& y_{\theta,k} \in \{0,1\}, && \theta \in \Theta \setminus \{|T|\},\, k \in K. && (6)
\end{aligned}$$

Clearly, the objective function minimizes the total busy time of all servers. Constraints (3) make sure that every item is scheduled precisely once, whereas Inequalities (4) prevent more than g items from entering a server at the same time. Conversely, an inactive server cannot accommodate any item.

Remark 14. *The above version of the assignment model follows the presentation chosen in [23]. In fact, if only C (instead of g) is given explicitly, the cardinality conditions (4) may also be replaced by classic capacity constraints of type*

$$\sum_{i \in I_{t_\theta}} c_i \cdot x_{ik} \leq C \cdot y_{\theta,k} \tag{7}$$

for any $\theta \in \Theta \setminus \{|T|\}$ and $k \in K$. In our computational tests, this modification will be particularly important for the instances of Category (A) not belonging to the unit-size case. In contrast, we decided to keep the original formulation from [23] in the theoretical discussion following below. Please note that the results deduced in this section can straightforwardly be transferred to an assignment model with (7) instead of (4).

As always, models of Kantorovich type possess some serious drawbacks. At first, we are witnessing a large degree of symmetry arising from permutations on the set K. Moreover, the LP bound is also rather weak as can be seen in the following result.

Theorem 15 (see Proposition 3.2 in [23]). *The additive integrality gap $z^{ass,\star} - z_{LP}^{ass,\star}$ is unbounded.*

To improve the assignment-based formulation, the following techniques (known from ILP models for the TBPP) will be applied:

- Obviously, the assignment variables are only required for $(i,k) \in \Delta$, where Δ just contains the index pairs with $i \geq k$. As a consequence of that, not every item can be assigned to every server anymore, so that server-dependent time sets $\Theta(k)$ can be used to reduce the set of y-variables, too.
- Let $\boldsymbol{\mathcal{C}} := \{\mathcal{C}_1, \ldots, \mathcal{C}_m\}$ denote the set of all maximum cliques of the interval graph (of a given instance E), and let $L := \{1, \ldots, m\}$ be a corresponding index set. For any maximal clique $\mathcal{C}_l$, we know that at least $\lceil |\mathcal{C}_l|/g \rceil$ servers are required to pack the items (in that specific clique). Hence, if $[t_{\theta(l)}, t_{\theta(l)+1})$ denotes a time interval belonging to $\mathcal{C}_l$, a

corresponding cut[6] is given by

$$\sum_{k\in K} y_{\theta(l),k} \geq \left\lceil \frac{|\mathcal{C}_l|}{g} \right\rceil. \tag{8}$$

Whenever the first reduction has already been applied, note that the summation on the left-hand side of (8) just has to contain those elements $k \in K$ for which $\theta(l) \in \Theta(k)$ is true.

- A heuristic solution can be passed to the solver. Given the results from Sect. 3, we will use BCH-T from Alg. 3 for that purpose.

Lemma 16. *The cut (8) is irrelevant if and only if* $|\mathcal{C}_l| \equiv 0 \mod g$ *holds for the considered clique.*

Proof. Let $\mathcal{C}_l \in \boldsymbol{\mathcal{C}}$ with $|\mathcal{C}_l| \equiv 0 \mod g$ be given. Then, there is some time interval $[t_{\theta(l)}, t_{\theta(l)+1})$ in which exactly the items from $\mathcal{C}_l$ are active. In particular, we also have $I_{t_{\theta(l)}} = \mathcal{C}_l$. Now, we obtain

$$|\mathcal{C}_l| = \left|I_{t_{\theta(l)}}\right| \overset{(3)}{=} \sum_{i\in I_{t_{\theta(l)}}} \sum_{k\in K} x_{ik} = \sum_{k\in K} \sum_{i\in I_{t_{\theta(l)}}} x_{ik} \overset{(4)}{\leq} \sum_{k\in K} g\cdot y_{\theta(l),k} = g\cdot \sum_{k\in K} y_{\theta(l),k}.$$

Since $|\mathcal{C}_l| \equiv 0 \mod g$, this is nothing else than (8). The reverse direction can be shown by similar arguments. □

As a consequence of Lemma 16, some of the valid inequalities listed in (8) are already contained in the ILP model. Hence, we will only add this type of cuts for the cliques satisfying $|\mathcal{C}_l| \not\equiv 0 \mod g$. The resulting formulation will be considered as the state of the art in our computational experiments, and it will mainly be compared to a flow-based model introduced in the next subsection.

4.2. A New Exact Approach Using CAF Networks

In [21], an idea to transform an instance of temporal bin packing to a layer-based network with manageable (but still exponential) state space has been introduced. Effectively, the basic approach is to consider one layer per maximum clique of a given instance. Originally, see [12], the states appearing in these layers are defined as the possible bin fillings within the respective clique, and transitions between two states are added whenever the items contained in the considered states are compatible. Due to the very large resulting networks, this idea was not considered an efficient solution method for a long time. In this regard, the contribution of [21] consists in observing that these networks contains many "equivalent" states, in the sense that there are nodes having precisely the same incoming and precisely the same outgoing arcs. These nodes can be shrunk to one "representative" state leading to a much smaller graph size. In this new interpretation, the states of some layer l refer to the possible bin fillings "at the end of the respective clique" $\mathcal{C}_l$. Since the full details of the construction proposed in [21] are very technical and would require much more notation, we refer the interested reader to the original source at this point. For the sake of exposition, however, we devote ourselves to an example that should accurately present the basic construction steps of the graph even without explicit knowledge of the mathematical details (of such a construction).

[6]Note that the right-hand side of (8) is identical to the optimal value of a bin packing problem just using the items contained in $\mathcal{C}_l$. In fact, this optimal value has to be chosen on the right-hand side of (8) whenever (7) is used as the capacity condition.

Example 3. *Let us consider the unit-size instance E_1 with $g = C = 2$ described in Fig. 3. Its corresponding CAF network is displayed in Fig. 4.*

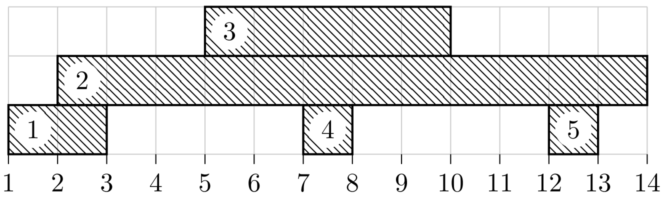

Figure 3: An illustration of E_1. This instance has three non-dominated starting times $T_S^{nd} = \{2, 7, 12\}$, and hence there are three maximum cliques (each containing the items active at the respective $t \in T_S^{nd}$) in the associated interval graph.

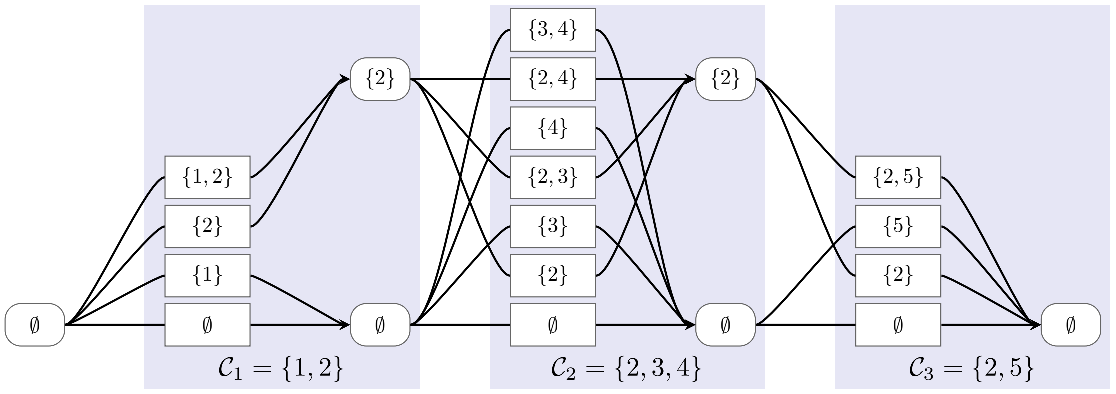

Figure 4: The CAF graph for E_1.

Nodes are given by oval shapes, whereas arcs are specified by connecting to nodes with a rectangular label (in the middle of the arc) specifying the items used when moving from one state to another. By way of example, let us note that "between" clique $\mathcal{C}_1$ and $\mathcal{C}_2$, the possible bin fillings are given by $\emptyset$ or $\{2\}$, since item $i = 1$ has already finished. Moreover, we highlight that item $i = 2$ is part of any clique. Hence, by way of example, a state containing item $i = 2$ cannot be connected with a state not containing the same item.

Fortunately, the overall structure of the network (originally developed for the TBPP) can also be used to address busy time minimization. The only thing to be aware of is that the costs of the arcs need to be changed. By that, also the composition of the objective function changes since any arc (and not only those starting in the source node of the artificial first layer) is important to obtain the correct objective value.

Since the decision made within a clique will hold for a certain time window, e.g., until the next clique starts, let us recall the definition

$$s(l) := \min \{s_i \, : \, i \in \mathcal{C}_l \setminus \mathcal{C}_{l-1}\},$$

from [21] indicating the earliest starting time of the items that are new in clique $l \in L$. In other terms, $s(l)$ can be interpreted as the "beginning" of clique $\mathcal{C}_l$. Note that $\mathcal{C}_0 := \emptyset$ is used in that definition to properly deal with $s(1)$. Moreover, we define an artificial boundary value $s(l+1) = \max\{e_i \, : \, i \in \mathcal{C}_m \setminus \mathcal{C}_{m-1}\}$, because the overall time horizon ends if all jobs have finished.

Based on these ingredients, at least two concepts of associating costs to an arc are reasonable:

(a) *clique-based costs* ("local view") meaning that the busy time units belonging to the decisions made within a clique $l \in L$ are localized, i.e., exclusively counted for the *responsibility interval*

$$R_l := [s(l), s(l+1))$$

of that particular clique. In other terms, the busy time of a pattern used in a clique is counted until the beginning of the next clique. Let $e = (l-1, l, J) \in \mathcal{E}_l$ represent an arc between layer $l-1$ and layer l using pattern $J \subseteq I$. For $J = \emptyset$, we define $\operatorname{cost}_{cb}(e) := 0$, because no item is assigned in that clique. Consequently, no busy time is recognized during interval R_l. For $J \neq \emptyset$, the costs are given by

$$\operatorname{cost}_{cb}(e) := \operatorname{span}\left(\bigcup_{i \in J} [s_i, e_i) \cap R_l\right). \tag{9}$$

By that, we measure the activity time of the items contained in the pattern, but only within the responsibility interval R_l of clique $\mathcal{C}_l$.

(b) *runtime-based costs* ("global view") meaning that the total costs (of a new decision) are globally counted at the moment the decision is made, even if some items from the chosen pattern J continue in the next clique(s). Again, for $J = \emptyset$ we set $\operatorname{cost}_{rb}(e) := 0$ because no item is placed. For $J \neq \emptyset$, the costs can be defined by

$$\operatorname{cost}_{rb}(e) := \operatorname{span}\left(\bigcup_{i \in J} [s_i, e_i) \setminus \bigcup_{i \in J \cap \mathcal{C}_{l-1}} [s_i, e_i)\right). \tag{10}$$

Since the costs of the items contained in the previous clique were already counted earlier, only the time instants added by the new items are important in this approach.

At the moment, computing the costs defined in (9) and (10) heavily depends on analyzing the span-operator applied to rather complicated sets. However, it is possible to find much easier closed-form terms for these objects.

Theorem 17. *Let $(l-1, l, J) \in \mathcal{E}_l$, $l \in L$, denote an arc appearing in the CAF network. Then, we have*

$$\operatorname{cost}_{cb}(e) = \begin{cases} \min\{s(l+1), \max_{i \in J} e_i\} - \max\{s(l), \min_{i \in J} s_i\}, & \text{if } J \neq \emptyset, \\ 0, & \text{if } J = \emptyset, \end{cases}$$

and

$$\operatorname{cost}_{rb}(e) = \begin{cases} \max_{i \in J} e_i - \max_{i \in J \cap \mathcal{C}_{l-1}} e_i, & \text{if } J \neq \emptyset,\ J \cap \mathcal{C}_{l-1} \neq \emptyset, \\ \max_{i \in J} e_i - \min_{i \in J} s_i, & \text{if } J \neq \emptyset,\ J \cap \mathcal{C}_{l-1} = \emptyset, \\ 0, & \text{if } J = \emptyset. \end{cases}$$

Proof. Let us treat both types of costs individually. Note that the case $J = \emptyset$ does not have to be considered, because its costs are already identical to the original definition.

(a) Let $J \neq \emptyset$ be given. Due to $J \subseteq \mathcal{C}_l$, all jobs of J share a nonempty joint activity interval. As a consequence of that, we have

$$\bigcup_{i \in J} [s_i, e_i) = \left[\min_{i \in J} s_i, \max_{i \in J} e_i\right). \tag{11}$$

However, this interval has to be restricted to the responsibility interval R_l of the currently considered clique. Note that, by definition, R_l and the interval appearing in (11) have a nonempty intersection. Hence, we obtain

$$\bigcup_{i \in J} [s_i, e_i) \cap R_l \overset{(11)}{=} \left[\min_{i \in J} s_i, \max_{i \in J} e_i\right) \cap [s(l), s(l+1))$$

$$= \left[\max\left\{s(l), \min_{i\in J} s_i\right\}, \min\left\{s(l+1), \max_{i\in J} e_i\right\}\right),$$

i.e., the argument of the span operator in (9) is a single interval. But then, the span and the length of that argument are the same by definition which proves the claim.

(b) Let us start with the most general case having $J \neq \emptyset$ and $J \cap \mathcal{C}_{l-1} \neq \emptyset$. At first, we note that

$$\bigcup_{i\in J}[s_i, e_i) = \left[\min_{i\in J} s_i, \max_{i\in J} e_i\right) \quad \text{and} \quad \bigcup_{i\in J\cap\mathcal{C}_{l-1}}[s_i, e_i) = \left[\min_{i\in J\cap\mathcal{C}_{l-1}} s_i, \max_{i\in J\cap\mathcal{C}_{l-1}} e_i\right) \tag{12}$$

are true, similar to the observations made in part (a) of the proof. Hence, our first step consists of rewriting the argument of the span operator in (10) as

$$\bigcup_{i\in J}[s_i, e_i) \setminus \bigcup_{i\in J\cap\mathcal{C}_{l-1}}[s_i, e_i) = \left[\min_{i\in J} s_i, \max_{i\in J} e_i\right) \setminus \left[\min_{i\in J\cap\mathcal{C}_{l-1}} s_i, \max_{i\in J\cap\mathcal{C}_{l-1}} e_i\right).$$

Now, observe that

$$\max_{i\in J\cap\mathcal{C}_{l-1}} e_i \leq \max_{i\in J} e_i, \qquad \min_{i\in J\cap\mathcal{C}_{l-1}} s_i = \min_{i\in J} s_i$$

hold. While the first is obvious (since the index sets of the maximum operators satisfy $J \cap \mathcal{C}_{l-1} \subseteq J$), the second one follows from the fact that $J \cap \mathcal{C}_{l-1} \neq \emptyset$ holds in the current case. To be more precise, any job $i \in J \setminus \mathcal{C}_{l-1}$ is new in that clique and, therefore, has a starting time $s_i \geq s(l)$. On the other hand, any job $i \in J \cap \mathcal{C}_{l-1}$ already started earlier than the current clique, meaning that $s_i < s(l)$ holds. Hence, the items from $J \setminus \mathcal{C}_{l-1}$ are not required for finding the minimum, and we have

$$\min_{i\in J\cap\mathcal{C}_{l-1}} s_i = \min_{i\in J} s_i.$$

For the sake of clarity, we highlight that this leads to

$$\left[\min_{i\in J} s_i, \max_{i\in J} e_i\right) \setminus \left[\min_{i\in J\cap\mathcal{C}_{l-1}} s_i, \max_{i\in J\cap\mathcal{C}_{l-1}} e_i\right) = \left[\max_{i\in J\cap\mathcal{C}_{l-1}} e_i, \max_{i\in J} e_i\right),$$

i.e., also the argument appearing in (10) actually is a single interval. Hence, its span is identical to its length, which is precisely the term appearing in the claim.

For $J \neq \emptyset$ and $J \cap \mathcal{C}_{l-1} = \emptyset$ note that, in fact, there is no set difference in (10). Hence, we have

$$\begin{aligned} \text{cost}_{rb}(e) &= \text{span}\left(\bigcup_{i\in J}[s_i, e_i)\right) \overset{(11)}{=} \text{span}\left(\left[\min_{i\in J} s_i, \max_{i\in J} e_i\right)\right) \\ &= \text{len}\left(\left[\min_{i\in J} s_i, \max_{i\in J} e_i\right)\right) = \max_{i\in J} e_i - \min_{i\in J} s_i, \end{aligned}$$

which completes the proof.

□

The previous theorem clarifies that, in both variants the costs of an arc can be easily computed. To better understand the differences between both methodologies, we first discuss an example before moving forward to the ILP model itself.

Example 4 (Continuation of Example 3)**.** *First of all, let us compute the responsibility intervals of any maximal clique:*

$$R_1 = [s(1), s(2)) = [1, 5), \quad R_2 = [s(2), s(3)) = [5, 12), \quad R_3 = [s(3), s(4)) = [12, 14).$$

Note that $s(4)$ is a hypothetical value (since there is no fourth clique) given by the latest time instant contained in T, which is the ending time of item $i = 2$. Now, when choosing the arc $e = (l-1, l, J) = (0, 1, \{1\})$, which models the placement of item $i = 1$ on an empty server (which becomes idle again directly after this item has finished), we have $J \cap \mathcal{C}_0 = \emptyset$ leading to

$$\text{cost}_{cb}(e) = \text{span}\left([1, 3) \cap R_1\right) = 2, \quad \text{cost}_{rb}(e) = \text{span}\left([1, 3)\right) = 2,$$

so both cost terms are equivalent. This is clear since item $i = 1$ is precisely contained in clique $l = 1$, and so the total costs of assigning this item are the costs this item causes in $\mathcal{C}_1$.

Alternatively, let us now consider the arc $e = (l-1, l, J) = (1, 2, \{2, 4\})$. Here, we have $J \cap \mathcal{C}_1 = \{2\}$, meaning that item $i = 2$ was already present in the previous clique. This observation is important for the runtime-based costs. Hence we obtain

$$\text{cost}_{cb}(e) = \text{span}\left([2, 14) \cap R_2\right) = 7, \quad \text{cost}_{rb}(e) = \text{span}\left([2, 14) \setminus [2, 14)\right) = 0.$$

Indeed, the responsibility interval of clique $l = 2$ was given by $R_2 = [5, 12)$. For the clique-based costs, scheduling items $\{2, 4\}$ means that the server is occupied during the entire interval R_2, so that seven units of busy time have to be counted. In the opposite case, the runtime-based costs of that arc are zero, since the costs of item $i = 2$ were already completely counted when the item appeared for the first time (i.e., when using arc $(0, 1, \{2\})$ or $(0, 1, \{1, 2\}$, one of which has to act as the predecessor of e in a feasible path).

As a consequence of the previous example, we see that the runtime-based concept (compared to the clique-based method) typically (i) assigns larger costs to arcs modeling the first appearance of an item, and (ii) may assign zero costs to some arcs, if some of its items were present in previous cliques. However, the total costs of an arbitrary but fixed path in the CAF graph are the same, no matter which of the two cost concepts was chosen. As a consequence of that, the LP bound does not depend on the choice of the cost paradigm. The latter was also observed in our internal test calculations not reported here. For the integer model, the different cost terms can, however, influence the variable selection for branching, and the success of heuristic methods applied by the solver. Hence, both these variants will be studied in the numerical tests.

Let us now define the flow-based ILP model in an abstract sense, meaning that we do not specify the choice of the cost terms. In analogy to [21], we use $\xi_e \in \mathbb{Z}_+$ to model the units of flow carried by arc $e = (l-1, l, J) \in \mathcal{E}_l$, $l \in L$. Moreover, the sets $\mathcal{E}(i)$ again gather all arcs indicating the positioning of item $i \in I$. Last but not least, we will use the abbreviations $\mathcal{E}^{in}$ and $\mathcal{E}^{out}$ to formulate the flow conservation constraints more conveniently. For the precise technical definitions, we again refer the reader to [21]. Overall we obtain the

Combinatorial Arcflow Model for the BTMP

$$z^{CAF} = \sum_{l \in L} \sum_{e \in \mathcal{E}_l} \text{cost}(e) \cdot \xi_e \to \min$$

$$\text{s.t.} \quad \sum_{e \in \mathcal{E}^{in}(l, \widetilde{J})} \xi_e = \sum_{e \in \mathcal{E}^{out}(l, \widetilde{J})} \xi_e, \qquad (l, \widetilde{J}) \in \mathcal{V} \setminus \{(0, \emptyset), (m, \emptyset)\}, \tag{13}$$

$$\sum_{e \in \mathcal{E}(i)} \xi_e = 1, \qquad i \in I, \tag{14}$$

$$\xi_e \in \mathbb{Z}_+, \qquad e \in \mathcal{E}_l, l \in L. \tag{15}$$

Obviously, the objective function minimizes the total busy time (i.e., the costs) of the configuration. Moreover, Constraints (13) ensure flow conservation at every interior vertex, whereas Conditions (14) guarantee that every item is used exactly once.

In analogy to Conditions (8), we can add valid inequalities[7] to CAF to raise the LP bound. Mathematically, this can be expressed by demanding

$$\sum_{e\in\mathcal{E}_l\setminus\{(l-1,l,\emptyset)\}} \xi_e \geq \left\lceil \frac{|\mathcal{C}_l|}{g} \right\rceil \tag{16}$$

for any $l \in L$. Due to Remark 5, the arc modeling the empty pattern does not have to be considered in that inequality. Again, we only have to consider maximal cliques satisfying $|\mathcal{C}_l| \not\equiv 0 \mod g$, since all other inequalities of that type are already implied by the model's constraints. Indeed, we obtain:

$$\begin{aligned} |\mathcal{C}_l| &\overset{(14)}{=} \sum_{i\in\mathcal{C}_l}\sum_{e\in\mathcal{E}(i)} \xi_e \overset{(17)}{=} \sum_{i\in\mathcal{C}_l}\sum_{e=(l-1,l,J)\in\mathcal{E}_l:i\in J} \xi_e = \sum_{e=(l-1,l,J)\in\mathcal{E}_l}\sum_{i\in J\cap\mathcal{C}_l} \xi_e \\ &= \sum_{e=(l-1,l,J)\in\mathcal{E}_l:J\neq\emptyset}\sum_{i\in J\cap\mathcal{C}_l} \xi_e = \sum_{e=(l-1,l,J)\in\mathcal{E}_l:J\neq\emptyset} \xi_e\cdot|J\cap\mathcal{C}_l| \\ &\leq \sum_{e=(l-1,l,J)\in\mathcal{E}_l:J\neq\emptyset} g\cdot\xi_e = g\cdot\sum_{e\in\mathcal{E}_l\setminus\{(l-1,l,\emptyset)\}} \xi_e. \end{aligned}$$

So, for $|\mathcal{C}_l| \equiv 0 \mod g$ the cut (16) does not contain any new information. In the previous calculation, we did use the fact that any pattern J appearing in an arc $(l-1,l,J)\in\mathcal{E}_l$ contains at most g items. Moreover, we highlight that any item $i \in \mathcal{C}_l$ has to start in a clique $l' \in L$ with $l' \leq l$, so that – by flow conservation – any flow related to $\mathcal{E}(i)$ has to traverse the layer l of CAF. This leads to

$$\sum_{e\in\mathcal{E}(i)} \xi_e = \sum_{e=(l-1,l,J)\in\mathcal{E}_l:i\in J} \xi_e. \tag{17}$$

The practical performance of CAF and the effects of the valid cuts will be studied in the numerical simulations appearing in the next subsection. However, one big advantage of the flow-based model, compared to the assignment formulation, is already given by the following result:

Theorem 18. *The LP bound of the flow-based model dominates the LP bound of the assignment model.*

Proof. The key idea of the proof is given by our the observation that the assignment model allows to reschedule the items between different time periods, whereas the decisions made in clique $\mathcal{C}_l$ of CAF will hold for the entire responsibility interval R_l (which typically is a collection of consecutive time periods). Hence, any feasible solution to the relaxed CAF model can be converted to a feasible point of the assignment model, but not vice versa. This proves the claim. □

4.3. The Savings Model: An Exact Approach for a Special Case

In this subsection, we consider the special case $g = C = 2$ which was recently motivated by the authors of [23] with reference to jobs appearing in machine learning applications conducted by Microsoft Azure. In this setting, a promising compact formulation, based on the concept of savings derived in Sect. 2, can be obtained. The general idea of such formulations was already

[7]If the unit-size case is not considered, these cuts have to be modified according to the instructions given for the assignment-based formulation before. In particular, the right-hand side of (16) will change to the optimal value of a bin packing problem just using the items from $\mathcal{C}_l$.

sketched in [23], so the contribution of this subsection is not necessarily to set up an improved version of the (partially prepared) model, but rather to derive it from and relate it to our flow model, thus providing a different perspective than the one used in the above article. Moreover, we will also present some new theoretical results and apply appropriate reductions.

4.3.1. Introduction and General Idea

On closer examination, the flow model introduced in the previous subsection offers some further reduction potential, whenever $g = C = 2$ is considered. Therefore, we start by presenting two important observations, referred to as (O1) and (O2), which are based on the following example.

Example 5. *Let us consider the instance E_2 and its corresponding CAF network visualized in Fig. 5 and Fig. 6, respectively.*

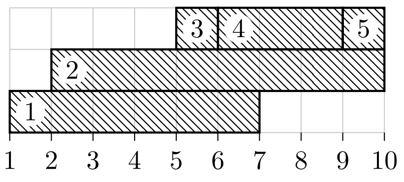

Figure 5: Visualization of the instance E_2.

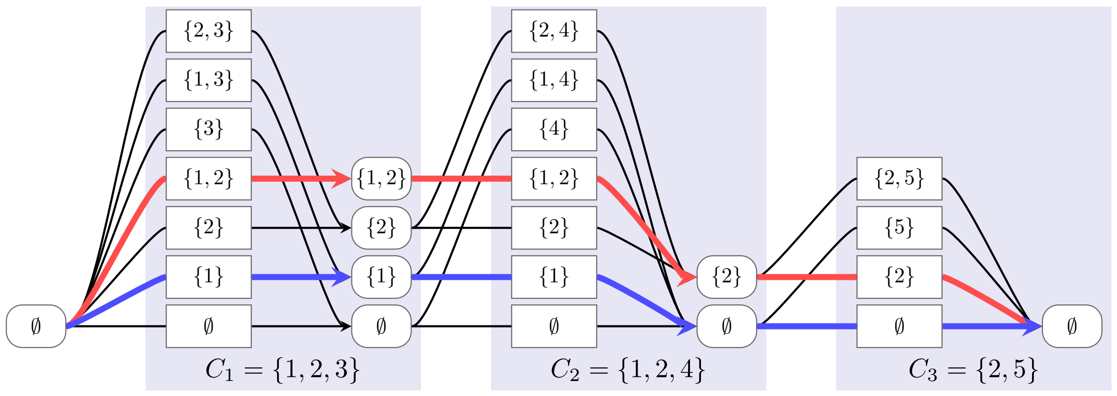

Figure 6: The CAF network for E_2. The paths painted blue or red will be required to exemplify the intended reduction rules.

.

(O1) Let us consider the red path appearing in Fig. 6. Obviously, the items $i = 1$ and $i = 2$ are both active within the first two cliques. In fact, for item pairs covering more than one clique, one variable would be sufficient since (for $g = C = 2$) no further item can be added to that configuration as long as both items are processed. However, CAF re-models pairing these jobs in any of the related cliques, although the decision has to be consistent across cliques. Consequently, the model contains some redundant variables in theory, but (for $g = C = 2$) these are identified with each other by flow conservation, since a state already containing two items cannot possess more than one incoming and outgoing arc. Note that, here we refrain from properly explaining how to formally implement this method in a graph-theoretical context. This is because we would require arcs spanning several cliques, and consequently the mathematical description of the CAF states and transitions would become even more technical.

(O2) In Theorem 6, we learned that maximizing the savings is an alternative to minimizing the busy time. So, in theory, it would also be possible to set up a flow model, whose arc costs correspond to the savings obtained by the pattern used (in the respective clique). By that, we would be able to remove those arcs, whose pattern label J consists of exactly one item. For the sake of exposition, note that the blue path illustrated in Fig. 6 would contain some redundant arcs, not representing any savings. However, removing those arcs may lead to additional source nodes in the graph, so the flow conservation has to be modified, in general. Moreover, Conditions (14) would have to be formulated with $\leq$ instead of $=$. Again, a correct formal description for the layer-based flow model turns out to be rather difficult, especially because the sets of incoming and outgoing arcs will be less structured. However, note that

Thus, we have seen that there are reasonable ideas to further simplify the CAF model for the BTMP, but their general description and concrete implementation would be relatively complicated. However, the previously obtained knowledge can be applied, at least for the case $g = C = 2$, in the context of a compact ILP formulation. For this purpose, we follow the guidelines in [23, Sect. 5] and define

$$o_{ij} := \operatorname{len}\left([s_i, e_i) \cap [s_j, e_j)\right)$$

for $i, j \in I$ with $i \neq j$. The term o_{ij} represents the temporal overlap[8] between the items i and j. Now, we let $\varrho_{ij} \in \{0,1\}$ denote whether jobs i and j are paired on the same server. Based on these ingredients, we obtain the

Savings Model for the BTMP (basic version)

$$z^{sav} = \operatorname{len}(I) - \frac{1}{2} \cdot \sum_{i \in I} \sum_{j \neq i} o_{ij} \cdot \varrho_{ij} \to \min$$

$$\text{s.t.} \quad \sum_{j \in \mathcal{C}_l} \varrho_{ij} \leq 1, \qquad l \in L, i \in \mathcal{C}_l, \tag{18}$$

$$\varrho_{ij} \in \{0,1\}, \qquad i \in I, j \in I \setminus \{i\}. \tag{19}$$

Effectively, this ILP formulation maximizes the total savings of an allocation in the objective function, and this is also the way it was introduced in [23]. However, for the sake of an easier comparison to other approaches, here this term is directly converted into the associated busy time via the calculation shown in the proof of Theorem 6. This is justified, because we know that the set of optimal solutions is the same for the two optimization goals.

Remark 19. *The additional factor $\frac{1}{2}$ in the objective function is required since the savings are double counted in this formulation. Indeed, this is caused by the symmetry contained in the model. More precisely, both pairs (i, j) and (j, i) are present in the index set of the ϱ-variables. Hence, in an optimal solution, both associated variables will always end up with the same value, counting the savings twice. In the remainder of this subsection, we will present an easy strategy to avoid the symmetries and the additional factor.*

Conditions (18) are responsible for just pairing two jobs together (so that they can be assigned to one server) in any maximal clique $\mathcal{C}_l \in \boldsymbol{\mathcal{C}}$. Moreover, the model does not explicitly capture those situations where a server is filled by just one single item (in some clique), because this situation is irrelevant for counting the savings. In fact, for a given clique $l \in L$ we can think of item $i \in \mathcal{C}_l$ being assigned to some server without any partner (in that clique), if and only if equality does not hold in Constraints (18). Due to the interpretation of finding pairs of items (i.e., pairs of nodes in the interval graph), this formulation is also referred to as the *matching formulation* in [23].

[8]In fact, for a subset $\{i, j\}$ containing just two items, this overlap is nothing else than the savings obtained by that item pair.

4.3.2. Reduction Methods and Theoretical Results

Let us now discuss some theoretical properties of the savings model suggested before. At first, we note that this formulation does not contain the server index k anymore, and so there is no symmetry arising from permutations. Moreover, the LP bound corresponding to the savings model dominates the bound resulting from the assignment model. This observation was contained in a relatively technical proof establishing a more detailed hierarchy of various lower bounds, see [23, Theorem 6.7]. However, the reasoning can be done much easier than reported therein:

Theorem 20. *We have $z_{LP}^{ass,\star} \leq z_{LP}^{sav,\star}$. Moreover, the LP bound of the savings model is identical to that of CAF.*

Proof. As derived earlier, the assignment-based model allows to reschedule the jobs in every interval $[t_\theta, t_{\theta+1})$, $\theta \in \Theta \setminus \{|T|\}$. Now, note that decisions made for a maximal clique $\mathcal{C}_l \in \boldsymbol{\mathcal{C}}$ typically hold for a contiguous sequence of more than one of these intervals. Hence, there is less variability in rearranging the items over time, and the claim is proved. The second statement follows directly from the fact that the savings model was deduced from CAF by applying Observations (O1) and (O2). $\square$

Hence, the savings model typically possesses some advantageous theoretical properties, namely a strong LP bound (compared to the assignment formulation) and a smaller model size (compared to CAF). However, the full potentials of this formulation have not been revealed yet by the contributions reported in [23]. This is not only because the model was not related to flow-based approaches in that article, but – more importantly – because it can be improved rather easily by at least two modifications:

(M1) At first, we do not have to cope with variables ϱ_{ij} the associated jobs of which do not overlap. Indeed, all these variables are multiplied by $o_{ij} = 0$ in the objective function, so that they can be omitted.

(M2) Secondly, there is no need to consider both pairs (i, j) and (j, i), because they model precisely the same decision, namely to pack jobs i and j to the same server. Hence, another straightforward reduction is to demand $j > i$ in any pair (i, j). As a consequence of that, double counting the savings in the objective function is avoided, and we can get rid of the factor $\frac{1}{2}$.

Typically, a large number of variables can be removed by these two modifications. Moreover, the number of nonzero elements in the constraint matrix becomes smaller, too. To implement (M1) and (M2), we define an item-specific set of potential partners

$$J(i) := \{j \in I \,|\, j > i,\, [s_i, e_i) \cap [s_j, e_j) \neq \emptyset\}$$

for any $i \in I$. Then, the reduced model is given by

Savings Model for the BTMP (reduced version)

$$z^{sav} = \operatorname{len}(I) - \sum_{i \in I} \sum_{j \in J(i)} o_{ij} \cdot \varrho_{ij} \to \min$$

$$\text{s.t.} \quad \sum_{j \in \mathcal{C}_l : j \in J(i) \vee i \in J(j)} \varrho_{ij} \leq 1, \qquad l \in L, i \in \mathcal{C}_l, \tag{20}$$

$$\varrho_{ij} \in \{0, 1\}, \qquad i \in I, j \in J(i). \tag{21}$$

Note that the index set for the summation in Constraints (20) looks a bit more complicated now. This is because job i can appear either as the first or as the second entry in an item pair, and we have to account for both possibilities (due to $j > i$) to state the conditions correctly. Moreover, at any position within the model, we could also replace $i \in I$ by $i \in I \setminus \{n\}$, since there is no possible partner j for the choice $i = n$ (as $J(n) = \emptyset$ holds by definition).

Of course, the cuts represented by (8) can also be added to the savings formulation. Given the fact that there is no explicit representation of a server just carrying one item, we have to demand

$$\sum_{i\in\mathcal{C}_l}\sum_{j\in J(i)\cap\mathcal{C}_l}\varrho_{ij} + \sum_{i\in\mathcal{C}_l}\left(1 - \sum_{j\in\mathcal{C}_l : j\in J(i)\vee i\in J(j)}\varrho_{ij}\right) \geq \left\lceil\frac{|\mathcal{C}_l|}{2}\right\rceil \tag{22}$$

for any $l \in L$. Of course, it is again sufficient to just consider those cliques with $|\mathcal{C}_l| \not\equiv 0 \mod 2$, i.e., the cliques with an odd number of items. Clearly, for those cliques, we end up with

$$\sum_{i\in\mathcal{C}_l}\sum_{j\in J(i)\cap\mathcal{C}_l}\varrho_{ij} + \sum_{i\in\mathcal{C}_l}\left(1 - \sum_{j\in\mathcal{C}_l : j\in J(i)\vee i\in J(j)}\varrho_{ij}\right) \geq \frac{|\mathcal{C}_l|+1}{2}. \tag{23}$$

Observe that the left-hand side of (22) or (23) consists of two sums representing the number of bins occupied by the items of clique $l \in L$. To be more precise, the first sum collects the servers accommodating two items at the same time, whereas the second sum collects the servers that just execute one single item. Remember that the correctness of interpreting the gap in Constraints (18) as the decision not to pair item i was explained earlier.

Obviously, the structure of these valid inequalities is rather tedious in the savings model. However, the following result shows that they can also be formulated in a more tractable way.

Lemma 21. *Let $l \in L$ be a clique with odd cardinality. Then, Condition (23) is equivalent to*

$$\sum_{i\in\mathcal{C}_l}\sum_{j\in J(i)\cap\mathcal{C}_l}\varrho_{ij} \leq \frac{|\mathcal{C}_l|-1}{2}. \tag{24}$$

Proof. Let $l \in L$ be a clique having odd cardinality. Then, the proof can be done by rearranging the terms in the following way:

$$\begin{aligned}
& \sum_{i\in\mathcal{C}_l}\sum_{j\in J(i)\cap\mathcal{C}_l}\varrho_{ij} + \sum_{i\in\mathcal{C}_l}\left(1 - \sum_{j\in\mathcal{C}_l : j\in J(i)\vee i\in J(j)}\varrho_{ij}\right) \geq \frac{|\mathcal{C}_l|+1}{2} \\
\iff & \sum_{i\in\mathcal{C}_l}\sum_{j\in J(i)\cap\mathcal{C}_l}\varrho_{ij} + |\mathcal{C}_l| - \sum_{i\in\mathcal{C}_l}\sum_{j\in\mathcal{C}_l : j\in J(i)\vee i\in J(j)}\varrho_{ij} \geq \frac{|\mathcal{C}_l|+1}{2} \\
\iff & \sum_{i\in\mathcal{C}_l}\sum_{j\in J(i)\cap\mathcal{C}_l}\varrho_{ij} - 2\sum_{i\in\mathcal{C}_l}\sum_{j\in J(i)\cap\mathcal{C}_l}\varrho_{ij} \geq -\frac{|\mathcal{C}_l|-1}{2} \\
\iff & \sum_{i\in\mathcal{C}_l}\sum_{j\in J(i)\cap\mathcal{C}_l}\varrho_{ij} \leq \frac{|\mathcal{C}_l|-1}{2}.
\end{aligned}$$

Note that we did use

$$\sum_{i\in\mathcal{C}_l}\sum_{j\in\mathcal{C}_l : j\in J(i)\vee i\in J(j)}\varrho_{ij} = 2\sum_{i\in\mathcal{C}_l}\sum_{j\in J(i)\cap\mathcal{C}_l}\varrho_{ij},$$

since any item pair (i, j) with $i \in \mathcal{C}_l$ and $j \in J(i) \cap \mathcal{C}_l$ appears precisely twice in the summation on the left-hand side. □

In fact, Conditions (24) are known from matching theory, see [10], and they basically state that there are at most $(|\mathcal{C}_l| - 1)/2$ edges in a matching having vertex set $\mathcal{C}_l$ with odd cardinality.

Overall, the savings model not only possesses the same LP bound like and (possibly) fewer variables than CAF, but also the same valid cuts can be applied to raise the LP bound. In fact, the only negative point is that we lose the typical structure of a flow formulation using equations (such as flow conservation) to link the variables. The effects of these opposing differences will be discussed in the next section.

Before that, however, we have to clarify the question of why a generalization to arbitrary values of $g \geq 3$ is not promising here. In fact, this is not necessarily related to the constraints themselves, but rather to the fact that the savings in the objective function can no longer be stated correctly without re-introducing various copies of (basically) the same decision to keep track of the temporal variation of the pattern. By that, the advantage of having fewer variables (compared to CAF) would be lost. More details will be explained in the following example.

Example 6. *Let us consider the unit-size instance E_3 depicted in Fig. 7 with $g = C = 3$.*

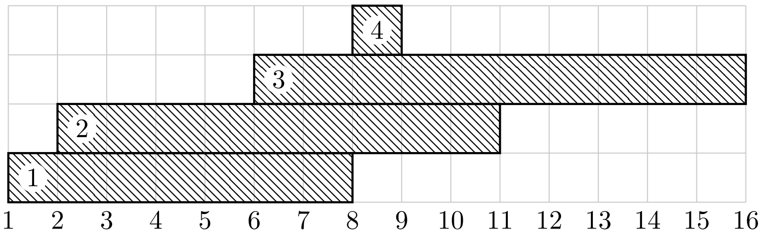

Figure 7: A visualization of E_3.

Obviously, this instance has two maximum cliques $\mathcal{C}_1 = \{1, 2, 3\}$ and $\mathcal{C}_2 = \{2, 3, 4\}$. In a straightforward extension of the savings model, using subsets $J \subseteq I$ in the indices, we would end up with the following decision variables:

$$\varrho_{\{1,2\}}, \varrho_{\{1,3\}}, \varrho_{\{2,3\}}, \varrho_{\{1,2,3\}}, \varrho_{\{2,4\}}, \varrho_{\{3,4\}}, \varrho_{\{2,3,4\}}.$$

The "problematic" feature of this instance is that there are two items ($i = 2$ and $i = 3$) being present in both cliques $l = 1$ and $l = 2$. In addition, these items can be processed together with item $i = 1$ in $\mathcal{C}_1$ and together with $i = 4$ in $\mathcal{C}_2$. Let us now compute the savings of these subsets:

$$\begin{aligned} \text{sav}(\{1,2,3\}) &= \text{len}(\{1,2,3\}) - \text{span}(\{1,2,3\}) = 26 - 15 = 11, \\ \text{sav}(\{2,3,4\}) &= \text{len}(\{2,3,4\}) - \text{span}(\{2,3,4\}) = 20 - 14 = 7. \end{aligned}$$

Obviously, choosing $\varrho_{1,2,3} = \varrho_{2,3,4} = 1$ would constitute an optimal solution. However, the additive objective function of the savings model would then lead to a maximum number of 18 units of savings (that is, a busy time of $\text{len}(I) - 18 = 27 - 18 = 9$ units), which is not correct. The reason is that some savings are double counted.

Note that it is even not possible to restrict the savings of a configurations $J \subseteq I$ with $|J| \leq g$ to the responsibility intervals R_l of all cliques containing that subpattern. Indeed, for $J = \{2, 3, 4\}$ just being part of $\mathcal{C}_2$ with $R_2 = [8, 16)$, this would result in $8+3+1-(3+1) = 8$ units of savings. But then, the total savings of a server just carrying these three items are not correct either.

In fact, to count the savings in the previous example correctly, we would have to know whether a server with $\varrho_{2,3,4} = 1$ already carried some other items before. Depending on whether $i = 1$ was part of the configuration in clique $\mathcal{C}_1$ or not, the contribution of $\varrho_{2,3,4} = 1$ in the objective function has to be different. Hence, the coefficients in the objective function of such a model would need to depend on other variables, so that they are not constant anymore. Or, alternatively, we would have to use different copies of the same decision variable. But this is, in fact, the general idea of CAF, so that for $g \geq 3$ no further advantages can be expected from using a matching-based formulation. Hence, extending the savings model to the more general case $g \geq 3$ does not lead to a more efficient solution approach compared to CAF.

5. Numerical Results

In this section, we compare the exact approaches presented before from a numerical perspective. To be more precise, we consider the instances from Category (A) and (C), introduced in Sect. 2, to account for the general case and the unit-size case, respectively. Any approach was coded in Python, and the ILP models were solved by Gurobi 10.0 with a time limit of 30 minutes on an AMD Ryzen 9 5900X processor with 64 GB RAM. For any of the considered ILP models, the solver is always given a warm start by providing the heuristic solution obtained from BCH-T, see Alg. 3. Our computational analysis will consist of two main parts:

(i) In a first experiment, we consider the instances of Category (A) not belonging to the unit-size case. As a consequence, the assignment-based model acts as the state-of-the-art formulation, and it will be compared to different variants of CAF. To be more precise, we will apply both cost structures (clique-based vs. runtime-based) with and without the valid inequalities conceptualized in (16). One aim of these investigations is to identify the best possible setup for CAF.

(ii) In a second experiment, we consider the instances of Category (C), meaning that we are dealing with the unit-size case. Hence, our comparison contains the assignment-based model, the "ideal" CAF setup (obtained from the first experiment), and the savings formulation. Also here, any model will be used in its maximally improved form, that is, with warm start and valid cuts.

Let us start with the results obtained for the first experiment. Here, the assignment-based formulation is compared with four variants of CAF, specified by two additional labels: 'CB' or 'RB' for clique-based or runtime-based costs, respectively, and '+ VI' if and only if the valid inequalities suggested by (16), in their modified form mentioned before, have been added.

The main insights given by our numerical experiment and the results listed in Tab. 3 are the following:

		CAF-RB		CAF-CB		CAF-RB + VI		CAF-CB + VI	
c_i	n	t	*opt*	t	*opt*	t	*opt*	t	*opt*
c_L	50	1.3	(20)	1.4	(20)	1.2	(20)	1.2	(20)
	100	795.7	(18)	704.2	(16)	599.6	(17)	599.4	(17)
	150	1800.0	(0)	1800.0	(0)	1618.2	(3)	1630.7	(4)
	200	1800.0	(0)	1800.0	(0)	1800.0	(0)	1800.0	(0)
	500	1800.0	(0)	1800.0	(0)	1800.0	(0)	1800.0	(0)
	1000	1800.0	(0)	1800.0	(0)	1800.0	(0)	1800.0	(0)
Average (Sum)		1332.8	(38)	1317.6	(36)	1269.8	(40)	**1271.9**	**(41)**
c_H	50	0.0	(20)	0.0	(20)	0.0	(20)	0.0	(20)
	100	0.6	(20)	0.6	(20)	0.6	(20)	0.7	(20)
	150	2.6	(20)	3.0	(20)	3.0	(20)	2.7	(20)
	200	105.3	(20)	108.1	(20)	82.1	(20)	77.1	(20)
	500	1101.6	(10)	1075.6	(11)	891.1	(11)	885.5	(11)
	1000	1581.5	(4)	1528.3	(5)	1316.6	(8)	1313.1	(8)
Average (Sum)		465.3	(94)	452.6	(96)	382.2	(99)	**379.8**	**(99)**
Total: Average (Sum)		899.0	(132)	885.1	(132)	826.0	(139)	**825.9**	**(140)**
Exit Gap		1.49%		1.43%		1.20%		**1.17%**	

Table 3: Numerical comparison of different CAF variants for instances of Category (A).

- At first, note that the assignment model (not tabulated here) was able to solve just one single instance (from the easiest class with $n = 50$ items) to proven optimality within the given time limit. Hence, any version of CAF is clearly better than the previous state of the art.

- Obviously, instances having small item sizes ('c_L') are more difficult to solve. In fact, any CAF version was able to solve more than 75% of the c_H-instances, but only roughly one third of the c_L-instances. In the latter case, there are much more feasible item combinations, meaning that the average number of states per clique is typically much larger, and so are the integer problems. As a consequence of that, already most of the instances with $n = 150$ cannot be solved in the c_L-scenario. This is a huge difference compared to the performance of CAF for the ordinary TBPP or the TBPP-FU, see [21]. However, we have to keep in mind that the values the objective function are much more heterogeneous[9] for busy time minimization, so that proving optimality of a feasible solution has become more difficult.

- Although the effects of the valid cuts are not very pronounced, it can be seen that they improve the performance of CAF to a moderate extent. Note that there seems to be a certain dichotomy among the instances considered: They are either solvable directly with any CAF version (see c_H) or too hard to be solved even by the improved variants (see c_L), since the model size is too large. Overall, the benefits of the valid cuts will become much more visible in our second experiment dealing with Category (C).

- For any of the listed approaches, the exit gap is very low, on average. Hence, we obtain good-quality feasible solutions, even if the point available when reaching the time limit was not proven to be optimal.

- As a last observation, we mention that there is no significant difference between the two cost structures. We attribute this to the fact that the internal routines of the solver are likely to anticipate the implications of clique-based costs[10]. When just comparing the setups with valid cuts, the clique-based costs ('CB') seem to be marginally better (in terms of optimal solutions obtained and the exit gap), but this is just a random effect which is due to precisely one additional c_L-instance with $n = 150$ solved to proven optimality right before the time limit would intervene. However, for the second experiment, we have to make a decision with respect to the "ideal" CAF setup, and so we will pass CAF-CB with valid cuts to the next numerical comparison.

In a second experiment, the unit-size instances of Category (C) will be studied computationally. At first, we mention that the assignment model is not able to solve a single instance from that benchmark set. Hence, in Tab. 4, we just include the results of two versions of CAF-CB, with and without valid cuts, and the savings model (also with valid cuts) tailored for the special case considered here. Again, all approaches are equipped with the feasible starting point produced by Alg. 3.

The main observations are given by:

[9]By that, we mean the following: For the ordinary TBPP, the optimal value is typically not too large, so that many feasible allocations may possess the same objective value. For busy time minimization, the variety of different objective values is typically much larger.

[10]By that, we mean that fixing one variable will also fix other variables due to flow conservation. Hence, it may not make a difference whether the user itself directly calculates the global costs of a decision, or the solver comes to the same conclusion by using linking conditions among the variables.

n	λ	CAF-CB t	CAF-CB *opt*	CAF-CB + VI t	CAF-CB + VI *opt*	Savings Model t	Savings Model *opt*
200	5	0.8	(5)	0.1	(5)	0.0	(5)
	10	79.7	(5)	1.0	(5)	0.7	(5)
	20	1800.0	(0)	6.3	(5)	4.4	(5)
	40	1800.0	(0)	76.4	(5)	45.5	(5)
	80	1800.0	(0)	427.9	(5)	330.3	(5)
Average (Sum)		1096.1	(10)	102.3	(25)	**76.2**	**(25)**
400	5	4.1	(5)	0.2	(5)	0.1	(5)
	10	534.8	(5)	2.0	(5)	1.2	(5)
	20	1800.0	(0)	283.0	(5)	256.4	(5)
	40	1800.0	(0)	1800.0	(0)	1800.0	(0)
	80	1800.0	(0)	1800.0	(0)	1800.0	(0)
Average (Sum)		1187.8	(10)	777.1	(15)	**771.5**	**(15)**
600	5	9.9	(5)	0.5	(5)	0.1	(5)
	10	1440.0	(2)	12.9	(5)	7.8	(5)
	20	1800.0	(0)	1657.1	(1)	1572.9	(1)
	40	1800.0	(0)	1800.0	(0)	1800.0	(0)
	80	1800.0	(0)	1800.0	(0)	1800.0	(0)
Average (Sum)		1370.0	(7)	1054.1	(11)	**1036.2**	**(11)**
800	5	14.6	(5)	0.7	(5)	0.2	(5)
	10	1742.1	(1)	26.1	(5)	10.4	(5)
	20	1800.0	(0)	1800.0	(0)	1800.0	(0)
Average (Sum)		1185.6	(6)	608.9	(10)	**603.5**	**(10)**
1200	5	50.8	(5)	1.2	(5)	0.2	(5)
	10	1800.0	(0)	76.9	(5)	33.6	(5)
	20	1800.0	(0)	1800.0	(0)	1800.0	(0)
Average (Sum)		1216.9	(5)	626.0	(10)	**611.3**	**(10)**
1600	5	72.5	(5)	2.5	(5)	0.3	(5)
	10	1800.0	(0)	615.6	(5)	204.3	(5)
	20	1800.0	(0)	1800.0	(0)	1800.0	(0)
Average (Sum)		1224.2	(5)	806.0	(10)	**668.2**	**(10)**
2000	5	190.9	(5)	3.4	(5)	**0.4**	**(5)**
	10	1800.0	(0)	1325.1	(3)	560.9	(5)
	20	1800.0	(0)	1800.0	(0)	1800.0	(0)
Average (Sum)		1263.6	(5)	1042.8	(8)	**787.1**	**(10)**
2400	5	218.7	(5)	6.1	(5)	0.5	(5)
	10	1800.0	(0)	1737.6	(1)	1206.1	(5)
	20	1800.0	(0)	1800.0	(0)	1800.0	(0)
Average (Sum)		1272.9	(5)	1181.2	(6)	**1002.2**	**(10)**
Total: Average (Sum)		1225.3	(53)	748.8	(95)	**681.2**	**(101)**
Exit Gap		0.58%		0.12%		**0.07%**	

Table 4: Numerical comparison of three approaches for Category (C)

- As expected, with increasing value of n or λ the instances become more difficult to solve. While this is obvious in the former case, note that the latter value manages the average lifetime of an item. Hence, a larger choice of λ leads to much more feasible item combinations (i.e., variables in the savings models). Similarly, the number and the average size of the cliques increase, so that CAF will become more difficult to handle.

- Adding valid cuts of type (16) considerably improves the performance of the flow model. Overall, almost 80% more instances are solved to proven optimal. Hence, the effect is much more pronounced than for instances of Category (A). We attribute this to the fact that for $|\mathcal{C}_l| \equiv 1 \mod 2$, the clique cuts always raise the lower bound by precisely 0.5 units of flow, which does not have to be the case for larger values of g. Moreover, the benchmark set (C) seems to contain many instances that may not be solvable by CAF-CB not just because of the pure model size, but since raising the bound is also difficult. At least the latter is supported by Conditions (16), so that there is much more success in dealing with "medium" instances, meaning that either n and/or λ is not too large.

- Although CAF possesses much more variables and constraints than the savings model[11], it is remarkable that the performance of both models is more or less the same for all instances with $n \leq 1600$. By that, we mean that the number of instances solved to proven optimality is always identical, but from a running time perspective, there are some slight advantages in favor of the (much smaller) savings model. Only for the very large scenarios with $n \geq 2000$ the latter formulation seems to benefit from the significantly reduced model size. However, the numbers collected in Tab. 4 clearly underline that the structure of CAF (following the classical guidelines of a flow model) seems to be more favorable for a general-purpose solver. Among others, the internal heuristics or cuts applied by Gurobi may be very familiar with handling flow conservation conditions. As a consequence of that, CAF is competitive with the state-of-the art, even in the special case $g = 2$. This is also supported by the fact that the average exit gap is almost negligeable for both formulations, so that the quality of the feasible solutions obtained is roughly equal, even if optimality could not be verified by the solver.

For completeness, the temporal development of all three performances is depicted in Fig. 8 (only (C1)) and Fig. 9 (complete Category (C)). These figures also highlight the key facts noted before, meaning that (i) adding valid cuts is very helpful for these special-case instances, and (ii) CAF is competitive with the savings model, although being much larger in size. The latter fact is also the reason why the difference between these two models is the largest at the beginning of the time axis in both figures. Overall, we have proved that CAF is a very powerful and more general approach for solving differently-characterized benchmark sets in busy time minimization.

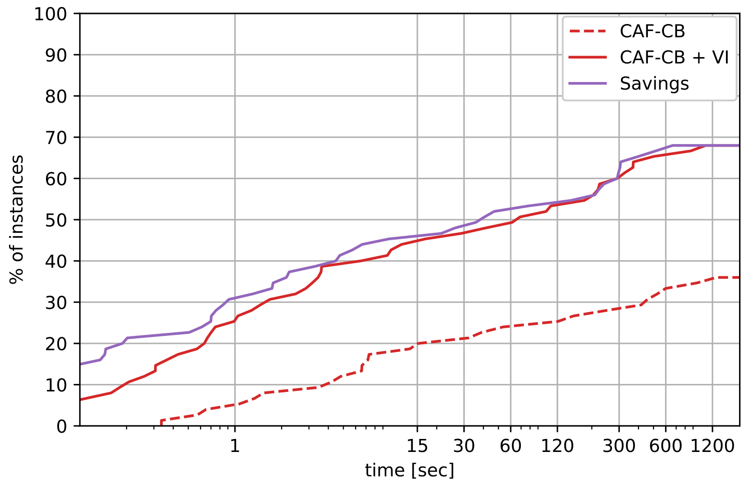

Figure 8: Performance profile for all instances from Category (C1), i.e., those with $n \leq 600$.

6. Conclusions

In this article, we studied the busy time minimization problem, an $\mathcal{NP}$-hard discrete optimization problem arising, among others, in job-to-server scheduling. After having introduced some

[11]This fact was already explained in the theoretical parts of this chapter, but numerical data further supporting this claim can be found in Tab. A.5 in Appendix A. Averaging over all instances from Category (C), more than 85% of the variables and constraints can be saved by moving from CAF to the savings model.

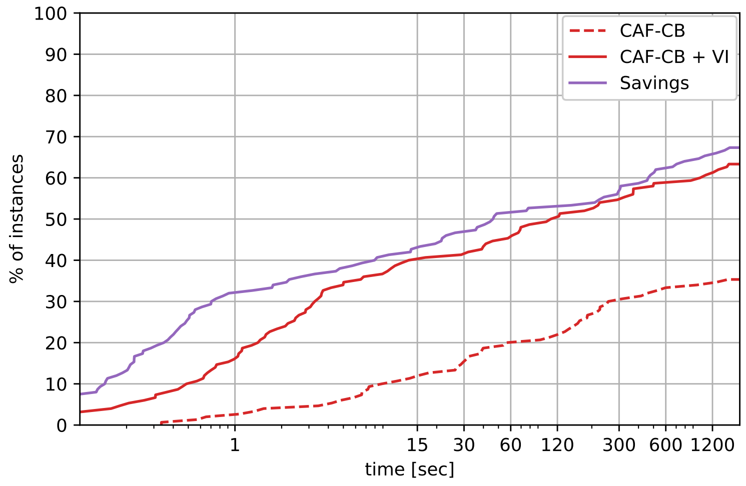

Figure 9: Performance profile for all instances from Category (C).

basic observations separating the BTMP from other closely related packing problems, we first gave an overview of heuristics from the literature and their associated worst-case performance factors. In this field, we proposed a new best-cost heuristic tackling some weak points of the aforementioned approaches. Remarkably, this heuristic performed better than the state-of-the-art for almost all benchmark instances from [2, 23], although not possessing a bounded worst-case behavior. In a second step, we investigated exact approaches and proposed a new combinatorial arcflow (CAF) model for the BTMP which is based on the constructions recently suggested in [21]. For this ILP model, two different cost paradigms as well as one family of valid inequalities are introduced and studied numerically. In the final calculations, it turns out that CAF is a very powerful solution approach for instances of general type, clearly beating the previous state-of-the-art. Furthermore, it is competitive with a matching-based approach, applicable to a very narrow special case only.

In terms of future research, some open questions related with the exact approximation factor of heuristic approaches have been collected throughout this work. Moreover, in contrast to ordinary temporal bin packing problems, the limits of CAF are obtained a bit faster for busy time minimization, so that there is still a decent amount of unsolved benchmark instances. Hence, there is some scientific need to further improve CAF or to consider branch-and-price based approaches.

CRediT authorship contribution statement

John Martinovic: Conceptualization; Methodology; Formal analysis; Writing - Original Draft

Nico Strasdat: Conceptualization; Methodology; Formal analysis; Software; Visualization

Conflict of Interest

The authors declare that they have no conflict of interest.

References

[1] Alicherry, M., Bhatia, R.: Line system design and a generalized coloring problem. Proceedings of the 11th Annual European Symposium on Algorithms (ESA), 19–30 (2003)

[2] 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)

[3] Biedl, T.: Graph-theoretic algorithms. Lecture Notes, University of Waterloo, Canada (2005)

[4] Chang, J., Khuller, S., Mukherjee, K.: LP rounding and combinatorial algorithms for minimizing active and busy time. Journal of Scheduling 20, 657–680 (2017)

[5] 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)

[6] 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)

[7] 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)

[8] de Lima V.L., Iori M., Miyazawa F.K.: Exact solution of network flow models with strong relaxations. Mathematical Programming 197, 813–846 (2023)

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

[10] Edmonds, J.: Maximum matching and a polyhedron with 0,1-vertices. Journal of Research of the National Bureau of Standards B 69, 125–130 (1965)

[11] Flammini, M., Monaco, G., Moscardelli, L., Shachnai, H., Shalom, M., Tamir, T., Zaks, S.: Minimizing total busy time in parallel scheduling with application to optical networks. Theoretical Computer Science 411, 3553–3562 (2010)

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

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

[14] Khandekar, R., Schieber, B., Shachnai, H., Tamir, T.: Minimizing Busy Time in Multiple Machine Real-time Scheduling. Proceedings of the International Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS), 169–180 (2010)

[15] Koomey, J.: Worldwide electricity used in data centers, Environmental Research Letters 3, 1–8 (2008)

[16] Korbacher, L., Irnich, S., Martinovic, J., Strasdat, N.: Solving the Skiving Stock Problem by a Combination of Stabilized Column Generation and the Reflect Arc-Flow Model. Discrete Applied Mathematics 334, 145-162 (2023)

[17] Kumar, V., Rudra, A.: Approximation Algorithms for Wavelength Assignment. Proceedings of the 25th Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS), 152–163 (2005)

[18] 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)

[19] Martinovic, J., Scheithauer, G., Valério de Carvalho, J.M.: A Comparative Study of the Arcflow Model and the One-Cut Model for one-dimensional Cutting Stock Problems. European Journal of Operational Research 266(2), 458–471 (2018)

[20] 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) (currently available online as: `https://optimization-online.org/2022/02/8791/`)

[21] Martinovic, J., Strasdat, N., Valério de Carvalho, J.M., Furini, F.: A Combinatorial Flow-based Formulation for Temporal Bin Packing Problems. European Journal of Operational Research 307(2), 554–574 (2023)

[22] Mertzios, G.B., Shalom, M., Voloshin, A., Wong, P.W.H., Zaks, S.: Optimizing busy time on parallel machines. Theoretical Computer Science 562, 524–541 (2015)

[23] Muir, C., Marshall, L., Toriello, A.: Temporal Bin Packing with Half-Capacity Jobs. Preprint Optimization Online (2022) (`https://optimization-online.org/?p=19925`)

[24] Olariu, S.: An optimal greedy heuristic to color interval graphs. Information Processing Letters 37, 21–25 (1991)

[25] Shalom, M., Voloshin, A., Wong, P.W.H., Yung, F.C.C, Zaks, S.: Online optimization of busy time on parallel machines. Theoretical Computer Science 560(2), 190–206 (2014)

[26] Tian, W., Yeo, C.S.: Minimizing total busy time in offline parallel scheduling with application to energy efficiency in cloud computing. Concurrency and Computation: Practice and Experience 27, 2470–3488 (2015)

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

[28] Winkler, P. and Zhang, L.: Wavelength Assignment and Generalized Interval Graph Coloring. Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms, Baltimore, Maryland, 830–831 (2003)

[29] Wolsey, L.A.: Valid Inequalities, Covering Problems and Discrete Dynamic Programs. Annals of Discrete Mathematics 1, 527–538 (1977)

Appendix A. Further Numerical Results

n	λ	CAF-CB + VI n_v	CAF-CB + VI n_c	Savings Model n_v	Savings Model n_c
200	5	2.8	2.0	1.1	0.7
	10	7.0	5.2	2.0	1.1
	20	18.6	15.2	3.5	1.7
	40	54.0	48.0	6.1	3.0
	80	127.3	117.3	10.1	4.0
Average (Sum)		41.9	37.5	4.6	2.1
400	5	6.0	4.2	2.2	1.4
	10	16.0	12.0	4.3	2.3
	20	47.0	39.3	8.0	4.0
	40	158.0	142.9	15.4	7.2
	80	411.9	386.9	25.3	11.1
Average (Sum)		127.8	117.1	11.0	5.2
600	5	8.8	6.0	3.3	2.1
	10	21.6	16.1	6.0	3.3
	20	67.5	56.3	11.8	5.9
	40	236.5	213.9	23.1	11.0
	80	703.0	662.1	41.3	18.2
Average (Sum)		207.5	190.9	17.1	8.1
800	5	11.5	7.9	4.3	2.8
	10	29.2	21.7	8.2	4.4
	20	96.2	80.7	16.2	8.2
Average (Sum)		45.7	36.8	9.6	5.1
1200	5	17.2	11.7	6.5	4.1
	10	45.6	34.3	12.3	6.9
	20	140.5	117.4	24.1	12.1
Average (Sum)		67.8	54.5	14.3	7.7
1600	5	23.2	15.8	8.7	5.5
	10	63.0	47.5	16.8	9.3
	20	185.9	155.3	31.9	16.1
Average (Sum)		90.7	72.9	19.2	10.3
2000	5	29.2	20.0	10.9	6.9
	10	76.7	57.6	20.8	11.4
	20	243.0	203.9	40.8	20.6
Average (Sum)		116.3	93.8	24.2	13.0
2400	5	34.8	23.7	13.1	8.3
	10	93.2	70.1	25.1	13.9
	20	283.6	237.4	48.3	24.5
Average (Sum)		137.2	110.4	28.8	15.5
Total: Average (Sum)		108.6	94.4	15.1	7.7

Table A.5: Model size for Category (C)