Bin Packing Problem with Time Lags

Orlando Rivera Letelier, François Clautiaux, Ruslan Sadykov

Bin Packing Problem with Time Lags

Orlando Rivera Letelier[1], François Clautiaux[2,3], and Ruslan Sadykov[3,2]

[1]Doctoral Program in Industrial Engineering and Operations Research, Universidad Adolfo Ibáñez, Av. Diagonal Las Torres 2640, Peñalolén, Santiago, Chile. 7941169
[2]IMB, Université de Bordeaux, 351 cours de la Libération, 33405 Talence, France
[3]Inria Bordeaux – Sud-ouest, 200 avenue de la Vieille Tour, 33405 Talence, France

November 3, 2020

Abstract

We introduce and motivate a variant of the bin packing problem where bins are assigned to time slots, and minimum and maximum lags are required between some pairs of items. We suggest two integer programming formulations for the problem: a compact one, and a stronger formulation with an exponential number of variables and constraints. We propose a branch-cut-and-price approach which exploits the latter formulation. For this purpose, we devise separation algorithms based on a mathematical characterization of feasible assignments for two important special cases of the problem. Computational experiments are reported for instances inspired from a real-case application of chemical treatment planning in vineyards, as well as for literature instances for special cases of the problem. The experimental results show the efficiency of our branch-cut-and-price approach, as it outperforms the compact formulation of newly proposed instances, and is able to obtain improved lower and upper bounds for literature instances.

1 Introduction

The bin packing problem is one of the most widely studied combinatorial optimization problems and has been known since the 1930s (Kantorovich, 1960). The classical bin packing problem (BPP) consists in assigning a set of weighted items to a minimum possible number of identical capacitated bins. Formally, we are given a set of items $V = \{1, \dots, n\}$, and an unlimited quantity of identical bins with a positive capacity W. Each item $i \in V$ has a positive weight $w_i \leq W$. The goal is to find a packing of items into the minimum number of bins, such that the total weight of items packed in each bin does not exceed its capacity.

In this paper, we introduce the *bin packing problem with time lags* (BPPTL) which is a generalization of the BPP. Here the assignment of items to bins is performed over a discretized time horizon, and there are generalized precedence relations between items. In the BPPTL, in addition to the set of items, we have a directed valued graph $G = (V, A, l)$ representing precedence constraints between items, as well as a positive integer value L which defines the upper bound on the number of bins which can be used in each time period. If the value L is large enough, *i.e* if in any feasible solution the number of bins is always sufficient at each time period, for instance $L \geq n$, we simply write $L = \infty$. Each arc $(i, j) \in A$ has an associated time lag $l_{i,j} \in \mathbb{Z}$, which can be either positive or negative. The BPPTL consists in assigning each item $i \in V$ to a pair $(b(i), p(i))$, with $b(i) \in \{1, \dots, L\}$ and $p(i) \in \mathbb{Z}_+$, such that bin capacity (1a)

and time lag (1b) constraints are satisfied:

$$\sum_{i \in V:\, b(i)=b, p(i)=p} w_i \leq W \quad \forall b \in \{1, \ldots, L\}, \forall p \in \mathbb{Z}_+, \tag{1a}$$

$$p(i) + l_{i,j} \leq p(j) \quad \forall (i,j) \in A. \tag{1b}$$

The objective is to find a feasible assignment that minimizes the number of bin-period pairs (b, p) which have at least one item assigned to them, or to determine that such an assignment does not exist.

Our motivation for studying the BPPTL stems from applications in which some tasks should be performed repeatedly using resources that are available on a pay-per-use basis. For example, a given task should be performed six times in a given time period. A possible way to deal with recurrent tasks is to determine a priori a fixed time between two consecutive occurrences of the same task. This over-constrained setting limits the sharing of ressources, and may lead to expensive solutions. On the other hand, in many applications the decision maker does not impose a fixed frequency of task occurrences. Only a desired frequency is specified, which may be altered to some extent if this leads to a cost reduction. Thus, a more flexible approach is to impose positive and negative time lags between two consecutive occurrences of the same task. Positive lags determine the minimum elapsed time between two consecutive occurrences of a task, whereas negative lags determine the maximum elapsed time between them. This allows the decision maker to ensure a better usage of the resource. At the same time, solutions in which tasks are not distributed evenly over the time horizon are forbidden.

A specific case of such a problem can be observed when one performs the planning of phytosanitary treatments in a vineyard. The vineyard is divided into sectors, i.e. well-defined portions of the property that correspond to contiguous geographical areas. One is given a set of meta-requests for treatment, i.e. diseases against which the vineyard must be protected using phytosanitary products. For each appropriate sector-disease pair, a periodic treatment has to be performed. Each treatment consists of a sequence of tasks of a certain duration. A phytosanitary treatment protects the vines only for a certain period of time, so the same task has to be repeated over the time horizon. On the other hand, treatments should not be repeated too often due to regulations on the spread of phytosanitary products. Practically speaking, the same chemical product cannot be spread twice in a given period, whose length depends in the toxicity of the product. Therefore, two consecutive occurrences of the same treatment should be neither too far nor too close in time. Treatments are performed by identical rented vehicles with a fixed cost per day of usage. Vehicles are limited in the total duration of work in a day. A vehicle can perform several treatments in a day as long as their total duration does not exceed the maximum total duration. The time needed for a vehicle to go from one sector to another is negligible in comparison to the duration of treatments. Given the strict regulation on chemical product spreading, tasks are never fragmented and cannot be processed over two time periods.

This application can be modelled as the bin packing problem with time lags. Here, capacitated bins represent vehicles each having a maximum total duration. Weighted items represent treatment tasks of different durations. The time lags graph consists of double-linked chains, each chain representing one periodic treatment. Two consecutive items in the chain represent two consecutive occurrences of a phytosanitary treatment. There are two arcs between consecutive items: one with a positive time lag representing the minimum number of days between treatment occurrences, and the other with a negative time lag representing the maximum number of days between occurrences. Two special source and sink nodes together with arcs between them serve to fix the planning time horizon. An example of the graph is depicted in Figure 1. The optimal solution for this example is depicted in Figure 2. It so happens that this solution uses at most one bin per time period. In general, however, more than one bin can be used in the same period.

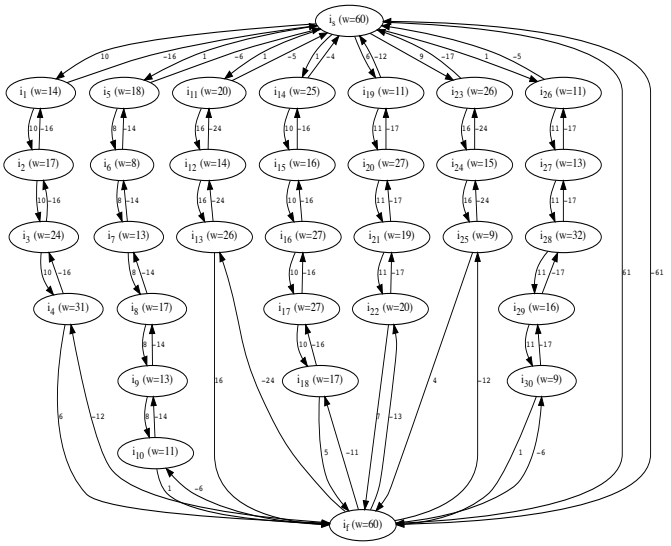

Figure 1: An example of the time lags graph for an instance of the phytosanitary treatments planning application

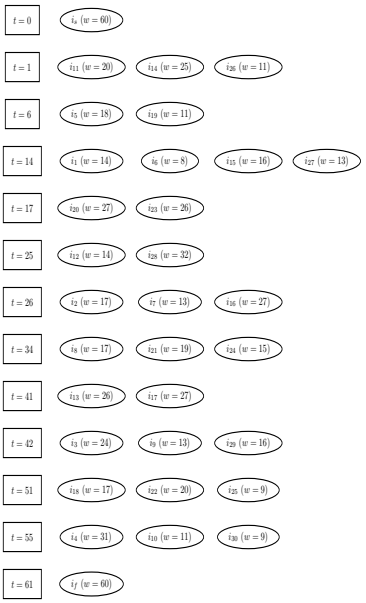

Figure 2: Solution of instance in Figure 1

1.1 Literature review

We now review the literature on the bin packing problems related to the BPPTL. The literature on the classical BPP is vast. Here we only present recent exact approaches to this problem. A comprehensive survey on exact methods for the BPP, including an original comparative computational study, was published by Delorme et al. (2016). After that, other exact specialized algorithms for the BPP were proposed by Delorme and Iori (2020) and by Wei et al. (2020). A generic BCP solver capable of solving exactly the BPP among other problems was proposed by Pessoa et al. (2020).

The BPPTL is closely related to some generalizations of the BPP known in the literature. The bin packing problem with precedence constraints (BPP-P) is a special case of the BPPTL in which all time lags are unitary and $L = 1$. The first exact approach for the BPP-P was developed by Dell'Amico et al. (2012). They proposed a large set of lower bounds, a variable neighborhood search, and a branch-and-bound algorithm. Later, Pereira (2016) suggested a dynamic programming-based heuristic and an exact enumeration procedure which uses several lower bounds and dominance rules to solve the BPP-P.

The simple assembly line balancing problem (SALBP-1) is a special case of the BPPTL in which all time lags are equal to zero and $L = 1$. The survey on simple assembly line balancing problems was presented by Scholl and Becker (2006). Morrison et al. (2014) proposed an exact branch-bound-and-remember algorithm for the SALBP-1. Pereira (2015) analyzed different families of lower bounds for the SALBP-1 and also presented new lower bounds which improved on the best known bounds for open instances of the problem.

Recently a generalization of both the BPP-P and the SALBP-1 was considered by Kramer et al. (2017). They introduced the bin packing problem with generalized precedence constraints (BPP-GP). In the BPP-GP considered by Kramer et al. (2017), time lags can only be non-negative, $L = 1$, and the objective is to minimize the makespan, i.e. the latest time period that has non-empty bins. As shown in Section 4, the number of used bins and the makespan objectives are equivalent for the case when $L = 1$ and every time lag is either one or zero. However, if some time lags are strictly greater than one, then the BPP-GP is not a special case of the BPPTL. In (Kramer et al., 2017), the authors proposed an effective iterated local search algorithm for the BPP-GP with the makespan objective. They also applied preprocessing techniques and lower

bounding procedures in order to estimate the quality of the solutions they obtained. Thus, many instances were solved to optimality.

Another related problem is the bin packing problem with conflicts (BPPC), in which a given undirected graph represents conflicts between items. Two adjacent items in the conflict graph cannot be put together in one bin. The objective function is the same as in the standard BPP, i.e. minimization of the number of bins used. The BPPC was introduced by Gendreau et al. (2004). Khanafer et al. (2010, 2012) proposed improved lower bounds for the BPPC based on dual-feasible functions, as well as tree-decomposition-based heuristics. Exact branch-and-price algorithms for the BPPC have been proposed in several papers (Fernandes Muritiba et al., 2010; Elhedhli et al., 2011; Sadykov and Vanderbeck, 2013; Wei et al., 2020).

In branch-and-price algorithms for the BPPC, the pricing problem is the knapsack problem with conflicts (KPC). This problem is interesting to us as it is a special case of the pricing problem considered in Section 3.2.2. The KPC has been solved either by MIP in (Fernandes Muritiba et al., 2010; Elhedhli et al., 2011), by dynamic programming and branch-and-bound algorithms in (Sadykov and Vanderbeck, 2013), or by a labelling algorithm as a resource-constrained shortest path problem in (Wei et al., 2020). Recently, improved combinatorial branch-and-bound algorithms for the KPC were proposed by Bettinelli et al. (2017) and Coniglio et al. (2020).

1.2 Contribution and outline

In this paper, as well as introducing and motivating the BPPTL, we present integer programming (IP) formulations for the problem. The first formulation is a compact one which involves one binary variable for every triple item-bin-time period. The second IP formulation with an exponential number of variables gives a relaxation to the BPPTL, similar to the one used in (Pereira, 2016) for the BPP-P. An exponential set of constraints is then introduced allowing us to turn the second formulation into an exact one for two important special cases of the BPPTL. The first one is when $L = \infty$. The second one is when $L = 1$, and time lags are non-negative. We present two ways of separating the introduced constraints: a MIP formulation and an enumeration algorithm. We also show how to take these constraints into account in the pricing problem, which turns into the knapsack problem with hard and soft conflicts (KPHSC), a generalization of the KPC. We formulate the KPHSC as a mixed integer program (MIP). Finally, for the two special cases of the BPPTL mentioned, we propose an exact branch-cut-and-price (BCP) algorithm which uses our cut separation routines, the MIP solver for solving the pricing problem which is the KPHSC, a strong diving primal heuristic, and the Ryan&Foster strong branching. We generate new instances for the BPPTL with $L = \infty$ inspired by the phytosanitary treatments planning application described above. We show that our BCP algorithm significantly outperforms a commercial MIP solver applied to the compact formulation for the problem. We also test our BCP algorithm on literature instances of the BPP-P, the SALBP-1, and the BPP-GP. We are able improve on the best known lower and upper bounds for numerous instances. Thus, optimality is reached for the majority of open instances.

The article is organized as follows. In Section 2 we present a compact formulation of the BPPTL, as well as a formulation with an exponential number of variables, which is a relaxation of the BPPTL. A BCP algorithm for the case $L = \infty$ is presented in Section 3. A modification of this algorithm for the case with $L = 1$ and non-negative time lags is given in Section 4. Computational results are presented in Section 5. We draw conclusions and discuss future work in Section 6.

2 Integer programming formulations

2.1 A compact formulation

Assume $T \in \mathbb{Z}_+$ is an upper bound to the number of time periods required to assign all the bins in an optimal solution. Denote the sets $\mathcal{L} = \{1, \ldots, L\}$ and $\mathcal{T} = \{1, \ldots, T\}$. Let binary variable $x_{i,b,p}$ take value one if and only if item $i \in V$ is assigned to bin (b, p) with $b \in \mathcal{L}$ and $p \in \mathcal{T}$. Also let binary variable $u_{b,p}$ take value one if only if bin (b, p) with $b \in \mathcal{L}$ and $p \in \mathcal{T}$ has at least one item assigned to it. The BPPTL can then be formulated as the following IP.

$$\min \sum_{b \in \mathcal{L}} \sum_{p \in \mathcal{T}} u_{b,p} \tag{2a}$$

$$\text{s.t.} \sum_{b \in \mathcal{L}} \sum_{p \in \mathcal{T}} x_{i,b,p} = 1 \qquad \forall i \in V, \tag{2b}$$

$$\sum_{i \in V} w_i x_{i,b,p} \leq W u_{b,p} \qquad \forall b \in \mathcal{L}, p \in \mathcal{T}, \tag{2c}$$

$$l_{i,j} + \sum_{p \in \mathcal{T}} p \cdot \sum_{b \in \mathcal{L}} x_{i,b,p} \leq \sum_{p \in \mathcal{T}} p \cdot \sum_{b \in \mathcal{L}} x_{j,b,p} \qquad \forall (i,j) \in A, \tag{2d}$$

$$x_{i,b,p} \in \{0,1\} \qquad \forall i \in V, b \in \mathcal{L}, p \in \mathcal{T}, \tag{2e}$$

$$u_{b,p} \in \{0,1\} \qquad \forall b \in \mathcal{J}, p \in \mathcal{T}. \tag{2f}$$

In this formulation, constraints (2b) require each item to be assigned to exactly one bin. Constraints (2c) impose two conditions: i) if an item is assigned to a bin (b, p) then the corresponding variable $u_{b,p}$ is equal to one; ii) the items assigned to a bin must have a cumulative weight smaller than or equal than the capacity of the bin. Constraints (2d) guarantee that the time lags are satisfied.

Some modifications to formulation (2) can be made. Constraints (2c) can be replaced by constraints giving a stronger linear relaxation:

$$\sum_{i \in V} w_i x_{i,b,p} \leq W \qquad \forall b \in \mathcal{L}, p \in \mathcal{T}, \tag{3a}$$

$$x_{i,b,p} \leq u_{b,p} \qquad \forall i \in V, b \in \mathcal{L}, p \in \mathcal{T}. \tag{3b}$$

These constraints should be used if some items have a weight equal to 0, since constraints (2c) would not count a bin in the objective value if all items assigned to it have weight 0.

Constraints (2d) can also be replaced by constraints which give a stronger linear relaxation:

$$\sum_{p'=1}^{p-l_{i,j}} \sum_{b \in \mathcal{L}} x_{i,b,p'} \geq \sum_{p'=1}^{p} \sum_{b \in \mathcal{L}} x_{j,b,p'} \qquad \forall (i,j) \in A, \forall p \in \mathcal{T} : p - l_{i,j} \leq T. \tag{4}$$

The number of constraints (4) is $\mathcal{O}(T|A|)$, while the number of constraints in (2d) is $\mathcal{O}(|A|)$. Thus, the stronger relaxation comes at the price of a larger formulation. The first use of constraints (4) in the context of a scheduling problem was made by Christofides et al. (1987). See (Artigues, 2017) for a more detailed discussion of these constraints.

Additionally, redundant constraints can be added to break symmetry. The bins used at each time period are equivalent: bins can be replaced without an impact on the feasibility and cost of the solution. Thus, without loss of generality we can impose that the bins with lower indices are chosen:

$$u_{b-1,p} \geq u_{b,p} \qquad \forall p \in \mathcal{T}, \forall b \in \mathcal{L} \setminus \{1\}. \tag{5}$$

Given a directed path in graph G from node i to node j, its length is the sum of the lags for the arcs in the path. Let $d(i,j)$ represent the length of the longest path from node i to node j in G. We convene $d(i,j) = -\infty$ if there is no path from i to j.

For each $i \in V$, let $es(i)$ represent the earliest time period in which i can be assigned to a bin, and let $ls(i)$ be the latest such time period. A natural pre-processing consists in fixing to zero all variables $x_{i,b,p}$ such that $i \in V$, $b \in \mathcal{L}$, and $p < es(i)$ or $p > ls(i)$. An approach to compute the values $es(i)$ and $ls(i)$ consists in adding two extra nodes to the graph G: the source node s fixed to artificial time period 0 and the sink node f fixed to artificial time period $T+1$. Then for each item $i \in V$ we add arcs (s,i) and (i,f) with time lag one. Then $es(i) = d(s,i)$ and $ls(i) = T + 1 - d(i,f)$.

2.2 An integer programming relaxation

The best exact approaches for the standard BPP are based on the extended formulation in which each variable represents a set of items packed into one bin. The main advantage of such a formulation is that its linear relaxation is stronger than that of compact formulations. For the BPPTL, there is no trivial way to come up with a similar extended formulation, since time lag constraints break the decomposable structure of the problem. Moreover, bins related to different time periods are no longer equivalent, and this symmetry cannot be exploited directly. However, it is possible to exploit the bin symmetry and the decomposable structure of a problem's relaxation. Therefore, we first formulate this relaxation of the BPPTL. Then, in the next two sections we will show how to turn this relaxation into an exact formulation for two special cases of the problem. This is done by defining an exponential family of constraints which cut off infeasible solutions of the relaxation.

In any feasible solution of the BPPTL, a set B of items assigned to the same bin must satisfy the following conditions:

$$\sum_{i \in B} w_i \leq W \tag{6a}$$

$$d(i,j) \leq 0 \text{ and } d(j,i) \leq 0 \qquad \forall i,j \in B. \tag{6b}$$

Constraint (6a) follows from (1a) and constraints (6b) follow from (1b). These two conditions state that the bin capacity constraint has to be satisfied, and that (transitive) lag constraints forbid some items from being packed in the same bin.

Let $\mathcal{B}$ be the collection of all possible non-empty sets B of items satisfying constraints (6a) and (6b). For each $B \in \mathcal{B}$, we define a binary variable λ_B taking value one if and only if set B of items occupies one bin in the solution. We denote by $\mathbb{1}_B$ the indicator function of B: $\mathbb{1}_B(i) = 1$ if $i \in B$, and $\mathbb{1}_B(i) = 0$ otherwise, for all $i \in V$. A relaxation of the BPPTL then can be formulated as follows.

$$\min \sum_{B \in \mathcal{B}} \lambda_B \tag{7a}$$

$$\text{s.t.} \sum_{B \in \mathcal{B}} \mathbb{1}_B(i) \lambda_B = 1 \qquad \forall i \in V, \tag{7b}$$

$$\lambda_B \in \{0,1\} \qquad \forall B \in \mathcal{B}. \tag{7c}$$

In this formulation the objective function is to minimize the number of bins selected in the solution. This relaxation has a similar structure to the one used for the simple assembly line balancing problem (Pereira, 2015) and the bin packing problem with precedences (Pereira, 2016), albeit with a different definition of $\mathcal{B}$. Moreover, integrality constraints (7c) are relaxed to the linear constraints $0 \leq \lambda_B \leq 1$ $\forall B \in \mathcal{B}$ in (Pereira, 2015, 2016).

A feasible solution $\bar{\lambda}$ for model (7) does not necessarily define a feasible solution for the BPPTL, since it is not always possible to assign bins in set $\{B \in \mathcal{B} : \bar{\lambda}_B = 1\}$ to time periods while satisfying the time lag constraints. This can be seen in the example illustrated in Figure 3. In this example, a feasible solution to (7) has items $\{A, B\}$ assigned to one bin and items $\{C, D\}$ to another. However, it is not possible to assign those bins to time periods satisfying the time lag constraints. In fact, item A must be assigned before D, and item C must be assigned before

B. One can see that the issue stems from the fact that when items are assigned to bins, lags between items become lags between bins, and cycles of positive length may appear. This concept of cycles between bins is formalized in Section 3, and is used to characterize infeasible solutions.

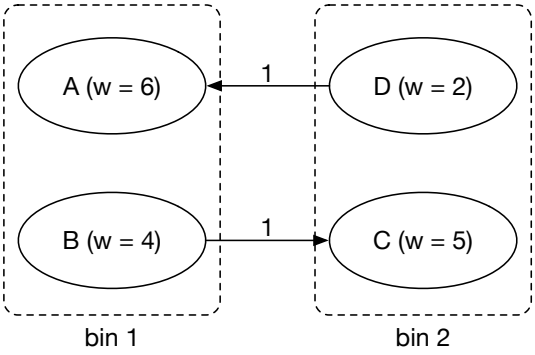

Figure 3: Example of instance in which a feasible solution to relaxation (7) does not define a feasible solution for the BPPTL. Here $W = 10$.

3 The case with unlimited number of available bins

In this section we assume that $L = \infty$, i.e. there is no restriction on the number of bins that can be assigned to a time period. We denote this variant of the problem as the BPPTL$^\infty$. We propose in this section a branch-cut-and-price algorithm based on relaxation 7 to solve this variant to optimality. In Section 3.1, we give a characterization of feasible solutions of the BPPTL$^\infty$ expressed by constraints involving only variables λ_B. Later, in Section 3.2, we describe the components of the branch-cut-and-price algorithm, i.e. separation algorithms for the cuts, IP formulation for the pricing problem, primal heuristic and branching.

3.1 Characterization of valid assignments

Let $\mathcal{P}$ be a partition of V. We denote by $\mathcal{P}(i)$ the element $B \in \mathcal{P}$ such that $i \in B$. For each $B, B' \in \mathcal{P}$, we define $\bar{d}(B, B') = \max\{d(i, j) : i \in B, j \in B'\}$ as the maximum distance (in terms of time lags) between an item in B and an item in B'. We introduce the notion of *aggregated graph*, which is useful for defining the necessary and sufficient conditions for a feasible solution to (7) to be feasible for the BPPTL.

Definition 1. Let $G = (V, A, l)$ be a directed valued graph, and $\mathcal{P}$ be a partition of V. The *aggregated graph* of G induced by $\mathcal{P}$ is the valued digraph $G^{\mathcal{P}} = (\mathcal{P}, A^{\mathcal{P}}, \bar{d})$, where $\mathcal{P}$ is the set of nodes in the graph, $A^{\mathcal{P}} := \left\{(B, B') \subseteq \mathcal{P} \times \mathcal{P} : \bar{d}(B, B') > -\infty\right\}$ is the set of arcs in the graph, and $\bar{d}(B, B')$ is the length of arc $(B, B') \in A^{\mathcal{P}}$.

We are going to show that whenever the aggregated graph induced by a partition is acyclic, the elements of the partition can be assigned to time periods satisfying the time lag constraints. But first, we need to prove the following lemma.

Lemma 1. *Let $G = (V, A, l)$ be a directed valued graph. Graph G has no cycle of positive length if and only if a mapping $\tau : V \longrightarrow \mathbb{Z}_+$ exists, such that for each $(i, h) \in A$, $\tau(i) + l_{i,h} \leq \tau(j)$.*

Proof. Proof. This proof is an adaptation of a similar result taken from (Bartusch et al., 1988). If such τ exists, consider any cycle with nodes $i_1, \ldots, i_K, i_{K+1} = i_1$. Then, for each $k \in \{1, \ldots, K\}$ we have $\tau(i_k) + l_{i_k, i_{k+1}} \leq \tau(i_{k+1})$. If we sum all these inequalities we get $\sum_{k=1}^{K} l_{i_k, i_{k+1}} \leq 0$, and then any cycle must have non-positive length.

Assume now that there is no cycle of positive length, and consider an extra *source* node $s \notin V$, and the graph $\bar{G} = (V \cup \{s\}, \bar{A}, \bar{l})$ such that $\bar{A} = A \cup \{(s,i) : i \in V\}$, and $\bar{l}_{i,j} = 1$ if $i = s$, and $\bar{l}_{i,j} = l_{i,j}$ otherwise.

For each $i \in V$, define $\tau(i)$ as the length of the longest path from s to i in $\bar{G}$. Since (G, A, l) has no cycle of positive length, then $(\bar{G}, \bar{A}, \bar{l})$ has no cycle of positive length, and then the longest path from s to i is a well defined finite number, taking a value of at least 1.

Finally, for each $(i,j) \in A$ it holds that $\tau(i) + l_{i,j} \leq \tau(j)$, since the length of the longest path from s to j cannot be smaller that the length of the longest path from s to i plus $l_{i,j}$. □

This lemma shows in particular that if the graph (G, A, l) has a cycle of positive length, there is no feasible solution for the instance.

Proposition 1. *Let $G = (V, A, l)$ be a directed valued graph and $\mathcal{P}$ be a partition of V. Also let $G^{\mathcal{P}} = (\mathcal{P}, A^{\mathcal{P}}, \bar{d})$ be the aggregated graph of G induced by $\mathcal{P}$. There is a mapping $\tau : \mathcal{P} \longrightarrow \mathbb{Z}_+$, such that for each $(i,j) \in A$, $\tau(\mathcal{P}(i)) + l_{i,j} \leq \tau(\mathcal{P}(j))$ if and only if there is no cycle of positive length in $G^{\mathcal{P}}$.*

Proof. Proof. Consider the mapping τ from Lemma 1 in the graph $G^{\mathcal{P}}$. Then for each $(B, B') \in A^{\mathcal{P}}$, it holds that $\tau(B) + \bar{d}(B, B') \leq \tau(B')$. But for each $i, j \in A$, we have $l_{i,j} \leq d(i,j)$, and since $\bar{d}(B, B') = \max\{d(i,j) : i \in B, j \in B'\}$, then $d(i,j) \leq \bar{d}(\mathcal{P}(i), \mathcal{P}(j))$. This implies that for each $(i,j) \in A$, $\tau(\mathcal{P}(i)) + l_{i,j} \leq \tau(\mathcal{P}(i)) + \bar{d}(\mathcal{P}(i), \mathcal{P}(j)) \leq \tau(\mathcal{P}(j))$. □

We call a partition $\mathcal{P} \subseteq \mathcal{B}$ *suitable* if graph $G^{\mathcal{P}}$ does not have a directed cycle of positive length, and we say that $\mathcal{P}$ is *unsuitable* otherwise. We denote by $\mathcal{N}$ the family of *unsuitable* partitions of V.

The partition of items induced by a feasible solution of an instance of the BPPTL$^\infty$ must be *suitable*, because the assignment of bins to time periods implies that there are no positive cycles in the aggregated graph. But Proposition 1 also shows that any *suitable* partition $\mathcal{P} \subseteq \mathcal{B}$ induces a feasible solution for the instance, and the proof also gives a procedure to recover the actual solution (the time of each bin) from the partition by computing the longest distance from an artificial source node to each node in the aggregated graph, which can be done in time $\mathcal{O}\left(|\mathcal{P}|\left|A^{\mathcal{P}}\right|\right)$.

With this characterization of a feasible solution, the following set of constraints can be added to formulation (7) to cut off infeasible solutions:

$$\sum_{B \in \mathcal{P}} \lambda_B \leq |\mathcal{P}| - 1 \qquad \forall \mathcal{P} \in \mathcal{N}. \tag{8}$$

Formulation (7) together with constraints (8) defines an exact formulation of the BPPTL$^\infty$. However, each constraint in (8) forbids only one *unsuitable* partition, and the coefficient of a variable λ_B in these constraints depends on set B. This makes dynamic generation of variables λ very difficult. We will now introduce an alternative to constraints (8) that defines a computationally tractable equivalent formulation.

The alternative constraints rely on the fact that cycles in the aggregated graphs are related to lags between pairs of items. Consider a partition $\mathcal{P}$ such that the induced aggregated graph $G^{\mathcal{P}}$ has a cycle of positive length $(B_1, B_2, \ldots, B_R)$, and denote $B_{R+1} = B_1$. For each arc (B_r, B_{r+1}) in the cycle there are elements $t_r \in B_r$ and $h_{r+1} \in B_{r+1}$ such that $\bar{d}(B_r, B_{r+1}) = d(t_r, h_{r+1})$. If we consider the set $\{(h_1, t_1), \ldots, (h_R, t_R)\}$ of pairs of items, any partition where each of these pairs is in the same bin induces an aggregated graph with a positive length cycle.

Figure 4 presents an example of a partition that induces a positive cycle in the aggregated graph. Any partition, in which pairs of nodes (h_1, t_1), (h_2, t_2) and (h_3, t_3) each belong to the same block of the partition, induces the aggregated graph with a positive length cycle.

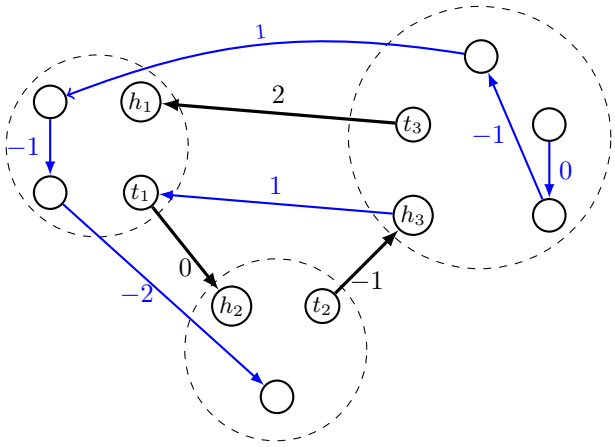

Figure 4: Example of partition that induces a positive cycle. Here $(h_1, h_2, h_3, t_1, t_2, t_3) \in \mathcal{C}$.

We now define the collection of all sets of pairs which induce a positive length cycle in the aggregated graph, and thus an infeasible solution.

Definition 2. For each $R \in \mathbb{N}$, $R \geq 2$, we define $\mathcal{C}_R \subseteq V^{2R}$ as the collection of all tuples $(h_1, \dots, h_R, t_1, \dots, t_R)$ that satisfy the following properties.

- All elements $h_1, \dots, h_R, t_1, \dots t_R$ are different.
- For each $r \in \{1, \dots, R\}$, $d(h_r, t_r) \leq 0$ and $d(t_r, h_r) \leq 0$.
- $\sum_{r=1}^{R} d(t_r, h_{r+1}) \geq 1$, where $h_{R+1} = h_1$.

We also denote $\mathcal{C} = \bigcup_{R \geq 2} \mathcal{C}_R$, and for each tuple $C = (h_1, \dots, h_R, t_1, \dots, t_R) \in \mathcal{C}$ we define as $\mathcal{F}_C$ the set of pairs of items which constitute C: $\mathcal{F}_C = \big\{\{h_1, t_1\}, \{h_2, t_2\}, \dots, \{h_R, t_R\}\big\}$.

We can now characterize an *unsuitable* partition as a partition that contains a forbidden set of pairs.

Proposition 2. *A partition $\mathcal{P} \subseteq \mathcal{B}$ is* unsuitable *if and only if $C \in \mathcal{C}$ exists, such that for each pair $\{h_r, t_r\} \in \mathcal{F}_C$, $\mathcal{P}$ contains an element B_r containing both h_r and t_r: $\{h_r, t_r\} \subseteq B_r \in \mathcal{P}$.*

Proof. Proof. Let $\mathcal{P} \subseteq \mathcal{B}$ be a partition of V, and let $C \in \mathcal{C}_R$, $R \geq 2$, be such that each $\{h_r, t_r\} \in \mathcal{F}_C$ is contained in $B_r \in \mathcal{P}$ for each $r \in \{1, \dots, R\}$. Then, by definition of $\bar{d}$, for each $r \in \{1, \dots, R\}$ we have $\bar{d}(B_r, B_{r+1}) \geq d(t_r, h_{r+1})$, where $B_{R+1} = B_1$. This implies that $\sum_{r=1}^{R} \bar{d}(B_r, B_{r+1}) \geq 1$, and then there is a circuit in $(G^{\mathcal{P}}, \bar{d})$ of positive length. Since the circuit can be decomposed into multiple cycles, and the length of the circuit is equal to the sum of the length of those cycles, one of those cycles must be of positive length, and then $\mathcal{P}$ must be *unsuitable*.

Assume now that $\mathcal{P}$ is *unsuitable*, and consider a cycle of positive length in graph $G^{\mathcal{P}}$ with the minimum possible number of nodes. Let R be the number of nodes of that cycle, and let the nodes of the cycle be $B_1, B_2, \dots, B_R$.

For each arc (B_r, B_{r+1}) in this cycle, let $t_r \in B_r$ and $h_{r+1} \in B_{r+1}$ be such that $d(t_r, h_{r+1}) = \bar{d}(B_r, B_{r+1})$, where $B_{R+1} = B_1$. Thus we have $\{h_r, t_r\} \subseteq B_r$, and since the cycle in the aggregated graph has positive length, we have $\sum_{r=1}^{R} d(t_r, h_{r+1}) = \sum_{r=1}^{R} \bar{d}(B_r, B_{r+1}) \geq 1$.

We now show that all these elements $(h_1, \ldots, t_R)$ are different. Since all B_r are different elements of a partition of V, for each $r, r' \in \{1, \ldots, R\}$ with $r \neq r'$ it holds $h_r \neq h_{r'}$, $t_r \neq t_{r'}$, and $h_r \neq t_{r'}$. If $h_r = t_r$ then $\bar{d}(B_{r-1}, B_{r+1}) \geq d(t_{r-1}, h_r) + d(t_r, h_{r+1}) = \bar{d}(B_{r-1}, B_r) + \bar{d}(B_r, B_{r+1})$, and then removing B_r from the cycle would still give a cycle of positive length, which contradicts the minimality of the cycle considered. Then, all the elements $h_1, \ldots, t_R$ are different.

Finally, since $B_r \in \mathcal{B}$, then $d(h_r, t_r) \leq 0$ and $d(t_r, h_r) \leq 0$. □

The following reformulation is valid for the BPPTL$^\infty$ due to Proposition 2.

$$\min \sum_{B \in \mathcal{B}} \lambda_B \tag{9a}$$

$$\text{s.t.} \sum_{B \in \mathcal{B}} \mathbb{1}_B(i) \lambda_B = 1 \qquad \forall i \in V, \tag{9b}$$

$$\sum_{r=1}^{R} \sum_{\substack{B \in \mathcal{B}, \\ \{h_r, t_r\} \subseteq B}} \lambda_B \leq R - 1 \qquad \forall R \geq 2, \forall (h_1, \ldots, t_R) \in \mathcal{C}_R, \tag{9c}$$

$$\lambda_B \in \{0, 1\} \qquad \forall B \in \mathcal{B}. \tag{9d}$$

3.2 The branch-cut-and-price algorithm

We develop a branch-cut-and-price approach to solve formulation (9). At each node of the branch-and-bound tree the linear relaxation is solved by the following column and cut generation procedure.

- *Step 0:* Start with an initial set of variables λ and constraints (9c).
- *Step 1:* Solve the restricted master problem (RMP). The RMP is the linear relaxation of (9) with the current restricted subset of columns and constraints.
- *Step 2:* Solve the pricing problem which looks for a variable λ_B with $B \in \mathcal{B}$ with negative reduced cost. If such a column is found, add λ_B to the RMP and go back to *Step 1*.
- *Step 3:* Solve the separation problem. The separation problem consists in finding a constraint (9c) violated by the current solution of the RMP.

Approaches for solving the separation problem are presented in Section 3.2.1. The MIP formulation for the pricing problem is proposed in Section 3.2.2. Other components of the BCP algorithm are described in Section 3.2.3.

3.2.1 Separation.

Given a solution $\bar{\lambda} = \left(\bar{\lambda}_B\right)_{B \in \mathcal{B}}$ that satisfies (9b), we are looking for a constraint in (9c) that is violated by $\bar{\lambda}$.

If $\bar{\lambda}$ satisfies (9d), i.e. it is an integer solution, it induces a partition $\mathcal{P}$ of the items, and finding a violated constraint in (9c) is equivalent to finding a cycle of positive length in the aggregated graph $G^{\mathcal{P}}$, according to Proposition 2. We first construct the aggregated graph $G^{\mathcal{P}}$. Then we compute the longest distance for each pair of nodes in this graph, which can be done in time $\mathcal{O}\left(|\mathcal{P}|^3\right)$ with the Floyd-Warshall algorithm. Finally, we look for a cycle of a positive length by checking if the distance from any node to itself is positive. In the event that a cycle of positive length is found, the proof of Proposition 2 describes the procedure to find the tuple $(h_1, \ldots, t_R) \in \mathcal{C}$ and to generate the associated violated constraint.

Separating integer solutions is enough for the algorithm to be valid. However, in order to strengthen the relaxation, we also propose procedures to separate fractional solutions. To do so,

we define an auxiliary directed multigraph in which finding a cycle with some specific properties is equivalent to finding a violated constraint in (9c).

Let $\left(\bar{\lambda}_B\right)_{B\in\mathcal{B}}$ be a solution satisfying (9b) but not necessarily (9d). For each $i, j \in V$ with $i \neq j$ we define the value $\xi_{i,j} = \sum_{\substack{B\in\mathcal{B},\\ \{i,j\}\subseteq B}} \bar{\lambda}_B$. Then we construct multigraph $\mathcal{G} = (V, \mathcal{A})$ with two labels in each arc. Each arc $a \in \mathcal{A}$ is represented by a tuple $(h(a), t(a), l(a), v(a)) \in V \times V \times \mathbb{Z} \times \mathbb{R}$, where $h(a)$ is the head of the arc, and $t(a)$ is the tail of the arc. The set of arcs is composed of two subsets, $\mathcal{A} = \mathcal{A}_G \cup \mathcal{A}_\lambda$, where these subsets are defined as follows:

- For each $(i, j) \in A$, arc a in which $h(a) = i$, $t(a) = j$, $l(a) = d(i, j)$ and $v(a) = 0$ belongs to $\mathcal{A}_G$.
- For each pair $\{i, j\} \subseteq V$ such that $i \neq j$ and $\xi_{i,j} > 0$, arc a in which $h(a) = i$, $t(a) = j$, $l(a) = 0$, and $v(a) = 1 - \xi_{i,j}$ belongs to $\mathcal{A}_\lambda$.

There is one arc in $\mathcal{A}_G$ for each arc in the original graph G, and two arcs in $\mathcal{A}_\lambda$, one in each direction, for each pair of nodes that are contained in any $B \in \mathcal{B}$ such that $\bar{\lambda}_B > 0$. Note that in this graph there might be two arcs with the same head and tail, one of them in $\mathcal{A}_G$ and the other in $\mathcal{A}_\lambda$.

Proposition 3. *Let $\left(\bar{\lambda}_B\right)_{B\in\mathcal{B}}$ be a solution satisfying* (9b)*. Then it satisfies all constraints in* (9c) *if and only if multigraph $\mathcal{G} = (V, \mathcal{A})$ constructed from $\bar{\lambda}$ has no cycle $(a_1, a_2, \ldots, a_K)$ such that $\sum_{k=1}^{K} l(a_k) \geq 1$ and $\sum_{k=1}^{K} v(a_k) < 1$.*

Proof. Proof. ($\Rightarrow$) Assume first that $\bar{\lambda}$ violates a constraint in (9c), then there is $R \geq 2$ and $(h_1, \ldots, t_R) \in \mathcal{C}_R$ such that

$$\sum_{r=1}^{R} \sum_{\substack{B\in\mathcal{B},\\ \{h_r,t_r\}\subseteq B}} \bar{\lambda}_B > R - 1.$$

This is equivalent to

$$1 > \sum_{r=1}^{R} \left(1 - \sum_{\substack{B\in\mathcal{B},\\ \{h_r,t_r\}\subseteq B}} \bar{\lambda}_B \right) = \sum_{r=1}^{R} \left(1 - \xi_{h_r,t_r}\right).$$

With this we define the following cycle with $K = 2R$ arcs in graph $\mathcal{G}$. For each $r \in \{1, \ldots, R\}$, we take arc $a_{2r-1} \in \mathcal{A}_\lambda$ such that $t(a_{2r-1}) = h_r$ and $h(a_{2r-1}) = t_r$, and we take arc $a_{2r} \in \mathcal{A}_G$ such that $t(a_{2r}) = t_r$ and $h(a_{2r}) = h_{r+1}$. We have $v(a_{2r-1}) = 1 - \xi_{h_r,t_r}$ and $v(a_{2r}) = 0$. Summing over all arcs in the cycle, we obtain $\sum_{k=1}^{2R} v(a_k) < 1$. We also have $l(a_{2r-1}) = 0$ and $l(a_{2r}) = d(t_r, h_{r+1})$. Then by definition of $\mathcal{C}_R$ we have $\sum_{k=1}^{2R} l(a_k) = \sum_{r=1}^{R} d(t_r, h_{r+1}) \geq 1$.

($\Leftarrow$) Assume now that there are cycles $(a_1, a_2, \ldots, a_K)$ in graph $\mathcal{G}$ satisfying both $\sum_{k=1}^{K} l(a_k) \geq 1$ and $\sum_{k=1}^{K} v(a_k) < 1$. Among these cycles we take one with the minimum number K of arcs. We now show that the arcs in this cycle are alternating between arcs of $\mathcal{A}_G$ and $\mathcal{A}_\lambda$. Let a and a' be two consecutive arcs in the cycle, with $h(a) = i$, $t(a) = j$, $h(a') = j$ and $t(a') = k$. Now consider two cases.

i) If both arcs belong to $\mathcal{A}_G$, then we can replace both arcs by arc (i, k) in $\mathcal{A}_G$ since $d(i, k) \geq d(i, j) + d(j, k)$.

ii) If both arcs belong to $\mathcal{A}_\lambda$, then we have

$$\begin{aligned}
\xi_{i,j} + \xi_{j,k} &= \sum_{\substack{B\in\mathcal{B},\\ \{i,j\}\subseteq B}} \bar{\lambda}_B + \sum_{\substack{B\in\mathcal{B},\\ \{j,k\}\subseteq B}} \bar{\lambda}_B \\
&= \underbrace{\sum_{\substack{B\in\mathcal{B},\\ \{i,j\}\subseteq B, k\notin B}} \bar{\lambda}_B + \sum_{\substack{B\in\mathcal{B},\\ \{i,j\}\subseteq B, k\in B}} \bar{\lambda}_B + \sum_{\substack{B\in\mathcal{B},\\ \{j,k\}\subseteq B, i\notin B}} \bar{\lambda}_B}_{\leq \sum_{\substack{B\in\mathcal{B},\\ j\in B}} \bar{\lambda}_B} + \underbrace{\sum_{\substack{B\in\mathcal{B},\\ \{j,k\}\subseteq B, i\in B}} \bar{\lambda}_B}_{\leq \sum_{\substack{B\in\mathcal{B},\\ \{i,k\}\subseteq B}} \bar{\lambda}_B} \\
&\leq 1 + \xi_{i,k}.
\end{aligned}$$

The first consequence of this inequality is that $\xi_{i,k} > 0$, and arc (i,k) belongs to $\mathcal{A}_\lambda$, as otherwise we would have $v(a) + v(a') \geq 1$. The second consequence is that $v(a) + v(a') = 2 - \xi_{i,j} - \xi_{j,k} \geq 1 - \xi_{i,k}$, and value v for arc (i,k) is not larger than $v(a) + v(a')$. Then we can replace arcs a and a' by arc (i,k).

In both cases, there is a contradiction with the minimality of the number of arcs in the cycle. Thus we can conclude that the arcs are alternating between $\mathcal{A}_G$ and $\mathcal{A}_\lambda$. Thus K must be even, and we can denote $K = 2R$. Without loss of generality we assume $a_1 \in \mathcal{A}_\lambda$. We define $h_r = t(a_{2r-1})$ and $t_r = h(a_{2r-1})$ for each $r \in \{1,\dots,R\}$. Now we are going to show that $(h_1,\dots,t_R) \in \mathcal{C}_R$.

The minimality of the cycle implies that all nodes $(h_1,\dots,t_R)$ are different. Since arc (h_r,t_r) belongs to $\mathcal{A}_\lambda$, there must be some $B \in \mathcal{B}$ with $\bar{\lambda}_B > 0$ such that $\{h_r,t_r\} \subseteq B$. By definition of $\mathcal{B}$, this implies that $d(h_r,t_r) \leq 0$ and $d(t_r,h_r) \leq 0$. Finally, since $a_{2r-1} \in \mathcal{A}_\lambda$, $l(a_{2r-1}) = 0$, then $\sum_{r=1}^R l(a_{2r}) \geq 1$, which implies that $\sum_{r=1}^R d(t_r,h_{r+1}) \geq 1$. □

The first approach to find such a cycle in multigraph $\mathcal{G} = (V,\mathcal{A})$ is based on an IP. Let y_a be a binary variable which determines whether arc $a \in \mathcal{A}$ participates in the cycle or not. Consider the following IP formulation.

$$\min \sum_{a\in\mathcal{A}} v(a) y_a \tag{10a}$$

$$\text{s.t.} \sum_{a\in\mathcal{A}} l(a) y_a \geq 1, \tag{10b}$$

$$\sum_{a\in\mathcal{A}\,:\,h(a)=i} y_a = \sum_{a\in\mathcal{A}\,:\,t(a)=i} y_a \qquad \forall i \in V, \tag{10c}$$

$$y_a \in \{0,1\} \qquad \forall a \in \mathcal{A}. \tag{10d}$$

It can be seen that there is a cycle $(a_1,a_2,\dots,a_K)$ in $\mathcal{G}$ such that $\sum_{k=1}^K l(a_k) \geq 1$ and $\sum_{k=1}^K v(a_k) < 1$ if and only if the optimal value of (10) is strictly smaller than one. Constraints (10c) force the solution to be a collection of cycles. Due to constraints (10b), at least one cycle in this collection must have a positive sum of values l. If the objective function is strictly less than one, then any cycle in the collection has the sum of values v strictly less than one.

Generating several constraints in each separation round helps to improve convergence of the cut generation procedure. This can be achieved by solving formulation (10) multiple times. After each solving, a constraint is added to avoid selecting the same cut again. Suppose that cycle $(a_1,a_2,\dots,a_K)$ is found after solving the IP. Without loss of generality, let $a_1 \in \mathcal{A}_G$ and $R = 2K$. Then the constraint $\sum_{r=1}^R y_{a_{2r}} \leq R - 1$ is added to (10), which is resolved again.

We now describe the second approach to find a desired cycle in multigraph $\mathcal{G} = (V,\mathcal{A})$. Given a set A' of arcs, we denote $v(A') = \sum_{a\in A'} v(a)$ and $l(A') = \sum_{a\in A'} l(a)$, and given two nodes

$i, j \in V$ we denote $v_{i,j}$ the minimum possible sum of v values of a path from i to j. The approach consists in a partial enumeration of cycles C in $\mathcal{G}$ satisfying $v(C) < 1$ and $l(C) \geq 1$.

We perform one iteration for each node $i \in V$. In this iteration, we try to find cycles C containing node i such that $v(C) < 1$ and $l(C) \geq 1$. Then we mark node i to exclude cycles containing i from the search in subsequent iterations. In each iteration, we use a recursive procedure which takes the current partial path form node i to current node j, containing set C of arcs and set $V_C \cup \{j\}$ of nodes. This procedure iterates over arcs $a \in \delta^+(j)$. If head $h(a)$ of arc a has been marked, arc a is ignored. Otherwise, if node $h(a)$ is contained in V_C then a subset C' of the arcs in $C \cup \{a\}$ forms a cycle. In this case, if conditions $v(C') < 1$ and $l(C') \geq 1$ are held, then we store cycle C' inducing a violated inequality. If node $h(a)$ is neither blocked nor contained in V_C we check conditions $v(C) + v(a) + v(a') < 1$ and $l(C) + l(a) + l(a') \geq 1$, where $a' = (h(a), i)$. If none of these conditions is true, it is unlikely that arcs in $C \cup \{a\}$ are contained in a cycle inducing a valid inequality, and arc a is ignored. If both conditions are true, we augment the partial path with arc a and recursively call the same procedure. The separation algorithm by partial enumeration is formally presented in Algorithm 1.

```
Function DFSRecursion(i, j, V_C, C):
    for a ∈ δ⁺(j) do
        if h(a) is marked then continue
        if h(a) ∈ V_C then
            Find C' ⊆ C ∪ {a} that forms a cycle.
            if v(C') < 1 and l(C') ≥ 1 then store cycle C'
            continue
        end
        a' ⟵ (h(a), i)
        if v(C) + v(a) + v(a') ≥ 1 or l(C) + l(a) + l(a') ≤ 0 then continue
        DFSRecursion(i, h(a), V_C ∪ {j}, C ∪ {a})
    end
end

Function Main(G, v, l):
    for i ∈ V do
        DFSRecursion(i, i, ∅, ∅)
        Mark node i
    end
end
```

Algorithm 1: Fractional separation by partial enumeration

3.2.2 Pricing problem.

Consider an optimal dual solution $(\bar{\pi}, \bar{\mu})$ of the restricted master problem, where $(\pi_i)_{i \in V}$ are the dual variables of constraints (9b) and $(\mu_C)_{C \in \mathcal{C}}$ are the dual variables of constraints (9c).

The pricing problem consists in finding a set of items $B \in \mathcal{B}$ such that the reduced cost of variable λ_B is minimized. The coefficient of λ_B in the constraint (9b) corresponding to $i \in V$ is one if $i \in B$, and zero otherwise. The coefficient of λ_B in the constraint (9c) corresponding to tuple $C \in \mathcal{C}$ is equal to the number of item pairs $\{i, j\} \in \mathcal{F}_C$ such that both i and j are contained in B. We define $\theta(B, C) = |\{\{i, j\} \in \mathcal{F}_C : i \in B, j \in B\}|$. Then the reduced cost of variable λ_B is $1 - \sum_{i \in B} \bar{\pi}_i - \sum_{C \in \mathcal{C}} \theta(B, C)\bar{\mu}_C$.

We now describe a MIP to find a set B with minimum reduced cost. Let $\bar{\mathcal{C}}$ be the set of active constraints (9c): $\bar{\mathcal{C}} = \{C \in \mathcal{C} : \bar{\mu}_C < 0\}$. Also let $\bar{\mathcal{F}} = \bigcup_{C \in \bar{\mathcal{C}}} \mathcal{F}_C$. Let binary variable x_i be equal to one if item $i \in V$ belongs to B, and zero otherwise. Also let binary variable $y_{i,j}$

be equal to one if both items from pair $\{i,j\} \in \bar{\mathcal{F}}$ belong to B, and zero otherwise. Then the pricing problem can be formulated as the following MIP.

$$\max \sum_{i \in V} \bar{\pi}_i x_i + \sum_{\{i,j\} \in \bar{\mathcal{F}}} \left(\sum_{\substack{C \in \bar{\mathcal{C}}: \\ \{i,j\} \in \mathcal{F}_C}} \bar{\mu}_C \right) y_{i,j} - 1 \tag{11a}$$

$$\text{s.t.} \sum_{i \in V} w_i x_i \leq W, \tag{11b}$$

$$x_i + x_j \leq 1 \qquad \forall \{i,j\} \subseteq V, d(i,j) \geq 1 \text{ or } d(j,i) \geq 1, \tag{11c}$$

$$x_i + x_j \leq 1 + y_{i,j} \qquad \forall \{i,j\} \in \bar{\mathcal{F}}, \tag{11d}$$

$$x_i \in \{0,1\} \qquad \forall i \in V, \tag{11e}$$

$$y_{i,j} \geq 0 \qquad \forall \{i,j\} \in \bar{\mathcal{F}}. \tag{11f}$$

Given an optimal solution $(\bar{x}, \bar{y})$ to (11), we add to the restricted master problem variable $\lambda_{\bar{B}}$, where $\bar{B} = \{i \in V : \bar{x}_i = 1\}$. Constraints (11b) and (11c) require that $\bar{B} \in \mathcal{B}$. Since $\mu_C < 0$ for $C \in \bar{\mathcal{C}}$ and we are maximizing, each variable $\bar{y}_{i,j}$ takes the minimum possible value. Therefore, $\bar{y}_{i,j}$ is equal to one if and only if both $i \in \bar{B}$ and $j \in \bar{B}$. Therefore,

$$\sum_{\{i,j\} \in \bar{\mathcal{F}}} \sum_{\substack{C \in \bar{\mathcal{C}}: \\ \{i,j\} \in \mathcal{F}_C}} \mu_C \bar{y}_{i,j} = \sum_{C \in \mathcal{C}} \theta(B,C) \bar{\mu}_C,$$

and the objective value of solution $(\bar{x}, \bar{y})$ is equal to the reduced cost of $\lambda_{\bar{B}}$ with the opposite sign.

The pricing problem can be defined as the knapsack problem with hard and soft conflicts (KPHSC). Hard conflicts correspond to pairs $\{i,j\}$ such that either $d(i,j) \geq 1$ or $d(j,i) \geq 1$. Soft conflicts are pairs that are penalized for being both in the set, and correspond to pairs $\{i,j\} \in \bar{mF}$. The KPHSC generalizes the NP-hard knapsack problem with conflicts. The latter is NP-hard, and it is studied for example in (Sadykov and Vanderbeck, 2013; Bettinelli et al., 2017; Coniglio et al., 2020). To our knowledge, the KPHSC has not yet been studied in the literature.

3.2.3 Other components of the algorithm.

It is well known that the column generation approach may have convergence issues. To improve convergence, we use the automatic dual price smoothing stabilization technique proposed by Pessoa et al. (2018).

Having a good feasible solution is important for the efficiency of the BCP algorithm. Such a solution provides an upper bound for the optimal objective value, and thus many nodes in the branch-and-bound tree may be pruned by bound. To obtain feasible solutions, we use the strong diving heuristic proposed by Sadykov et al. (2019). We now briefly describe how the standard diving heuristic (Joncour et al., 2010) can be applied for the BPPTL and present its strong diving variant.

After solving a node in the branch-and-bound tree, we have a solution $\bar{\lambda}$ to the linear relaxation of formulation (9). If $\bar{\lambda}$ is integer, the node is pruned, otherwise we apply the diving heuristic. In this algorithm, we select a column λ_B such that its value $\bar{\lambda}_B$ in the fractional solution is the closest to 1. We then add this column to the partial solution and change the right-hand-side values of constraints (9b) corresponding to items in B to zero. Afterwards, the linear relaxation of (9) is resolved by column generation (without cut separation). This iterative process continues until the solution to the linear relaxation becomes integer or until the relaxation becomes infeasible.

In order to increase the efficiency of the diving heuristic, we apply the following problem-specific enhancement to it. Each time a column λ_B is fixed to one, we update lag $l_{i,j}$ to $\max\{l_{i,j}, 0\}$ for every pair $\{i, j\} \subseteq B$. Following this update, we recalculate values $d(i, j)$ and $d(j, i)$, for all pairs $\{i, j\} \subseteq V$, i.e. the lengths of the longest paths between each pair of nodes in the graph. If a value $d(i, j)$ becomes positive, the set of hard conflicts in the pricing problem is updated, i.e. constraint $x_i + x_j \leq 1$ is added to formulation (11), and columns λ_B such that $\{i, j\} \subseteq B$ are removed from the master problem.

We use the strong diving variant (Pessoa et al., 2018) of this diving heuristic. In this variant, we select up to 10 candidate columns λ_B, and we fix them temporarily to one, one-by-one, and resolve the master problem. Then we select the candidate column which resulted in the smallest increase in the lower bound obtained by solving the linear relaxation of formulation (9).

After applying the strong diving heuristic, if the primal-dual gap is positive, we perform branching. Here we use the Ryan and Foster branching (Ryan and Foster, 1981). To do so, we find a pair of items $\{i, j\} \subseteq V$ such that value $\sum_{B \in \mathcal{B}:\, \{i,j\} \subseteq B} \bar{\lambda}_B$ is fractional. Then we create two child nodes: in the first we impose the requirement that items i and j are put into the same bin, and in the second that items i and j are assigned to different bins. In both nodes, we add the corresponding constraints to the pricing problem: $x_i = x_j$ and $x_i + x_j \leq 1$. We also remove from the master problem columns which do not satisfy newly imposed constraints.

Choosing a pair of items to branch on is an important decision, which has a large impact on the efficiency of the BCP algorithm. To find better branching candidates, we use two-phase strong branching. In the first phase, we take up to 30 branching candidates (i.e. pairs of items), create child nodes for them, and solve only the restricted master problem for them without column and cut generation. Up to three best candidates are then chosen according to the product rule (Achterberg, 2007). For child nodes of these candidates, we perform full column and cut generation. The size of the branch-and-bound tree is estimated for them according to the approach suggested in (Kullmann, 2009). Then the branching candidate with the smallest estimated tree size is chosen.

4 The case with one available bin per time period and non-negative lags

If we consider the case in which only one bin can be used in each time period ($L = 1$), then the BCP algorithm proposed in Section 3 cannot be used. In this case, given a partition of the item set in bins, the problem of determining if those bins can result in a feasible solution is NP-complete, as shown by Finta and Liu (1996). For this reason, we limit ourselves to the case in which all lags are non-negative numbers. We denote this problem by BPPTL^1_+.

In this context, we can assume that the graph is acyclic. Otherwise it either has a cycle of positive length and then it is trivially infeasible, or it has a cycle with all the arcs having lag zero. In the latter case we can contract all the items in the cycle into one item and have an equivalent problem because these items must be assigned to the same time period, and thus to the same bin.

A problem related to the BPPTL^1_+ has been studied by Kramer et al. (2017), who consider the makespan objective function, i.e. minimizing the last time period in which any bin is used. If all time lags are either zero or one, it is equivalent to minimizing the makespan and the number of bins used. This case generalizes both the bin packing problem with precedence constraints, and the simple assembly line balancing problem, as shown in (Kramer et al., 2017). However, if some time lags are strictly larger than one, minimizing the makespan is not equivalent to the BPPTL^1_+, since a solution that is optimal for one objective function may be sub-optimal for the other objective function. An example of this can be seen in Figure 5. In this example, if we optimize the use of one of the two objective functions we get a solution that is sub-optimal for the other objective function. Minimizing the makespan gives a solution with makespan 5 that

uses 4 bins, but minimizing for the number of bins used gives a solution that uses 3 bins with makespan 6.

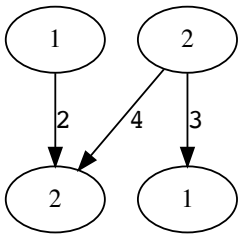

Figure 5: Example of instance with different optimal solutions for different objective functions. Here $W = 2$.

In this section, we show how to modify the BCP algorithm developed in Section 3 to solve to optimality the BPPTL$^1_+$.

4.1 Valid assignments

Any feasible solution for an instance of the BPPTL$^1_+$ is feasible for the BPPTL$^\infty$, and then the assignment of items to bins (which are also time periods in the BPPTL$^1_+$) must be composed of sets in $\mathcal{B}$, and must also satisfy constraints (9c). However, satisfying these constraints is not sufficient to induce a feasible solution for BPPTL$^1_+$, as can be seen in the example in Figure 6. In this example the time lags force each node to be assigned to the same time period, but the bin capacity does not allow one to assign all items to the same bin. Thus, there is a feasible solution to this instance of the BPPTL$^\infty$, but there are no feasible solutions for the corresponding instance of the BPPTL$^1_+$.

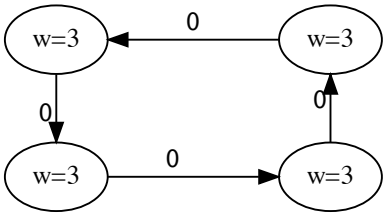

Figure 6: Example of instance with feasible solutions for $BPPTL^\infty$ but not for $BPPTL^1_+$. Capacity $W = 10$

In the following proposition, we characterize feasible solutions for the BPPTL$^1_+$ with a simple condition.

Proposition 4. *A partition $\mathcal{P} \subseteq \mathcal{B}$ induces a feasible solution to the $BPPTL^1_+$ if and only if the aggregated graph $G^{\mathcal{P}}$ is acyclic.*

Proof. Proof. Recall that any item set in $\mathcal{P}$ satisfies the capacity constraint. Therefore, only lag constraints have to be checked. If there is a feasible solution, then each set $B \in \mathcal{P}$ can be assigned to a time period p, and then the aggregated graph must be acyclic because otherwise the arc in the cycle with a tail assigned to the largest time period would be violated since all lags are non-negative. In the other direction, if $\mathcal{G}^{\mathcal{P}}$ is acyclic there is a topological order of the aggregated nodes. A feasible solution can be obtained by assigning bins to time periods following this topological order. □

In the proof, time lag values are not relevant. Then, any partition that induces a feasible solution also induces a feasible solution for any other graph with the same set of arcs, even with different (non-negative) lags. This implies that the time lag values are relevant just to define the conflicts for set $\mathcal{B}$. Thus it only matters if the value of each time lag is zero or a positive number. Therefore, all positive time lags can be replaced by unitary time lags without changing the set of optimal solutions.

In Section 3.1, we define set $\mathcal{C}_R$ to characterize partitions that induce an aggregated graph with a cycle of positive length, and thus an infeasible solution to the BPPTL$^{\infty}$. For the BPPTL$^{1}_{+}$, partitions that induce an aggregated graph with a zero length cycle are also infeasible. Analogously to Definition 3 for cycles of positive length, we now define the set of tuples which induce cycles of length 0 in the aggregated graph.

Definition 3. For each $R \in \mathbb{N}$, $R \geq 2$, let $\mathcal{C}^0_R \subseteq V^{2R}$ be the set of all tuples $(h_1, h_2, \ldots, h_R, t_1, \ldots, t_R)$ that satisfy the following properties.

- All the elements $h_1, \ldots, h_R, t_1, \ldots t_R$ are different.
- For each $r \in \{1, \ldots, R\}$, $d(h_r, t_r) \leq 0$ and $d(t_r, h_r) \leq 0$.
- $\sum_{r=1}^{R} d(t_r, h_{r+1}) = 0$, where $h_{R+1} = h_1$.

Recall the definition of value $\theta(B, C)$:

$$\theta(B, C) = \left|\left\{r \in \{1, \ldots, R\} \,:\, \{h_r, t_r\} \subseteq B\right\}\right|.$$

With this definition, constraints (9c) can be rewritten as:

$$\sum_{B \in \mathcal{B}} \theta(B, C)\lambda_B \leq R - 1 \qquad \forall R \geq 2, \forall C \in \mathcal{C}_R. \tag{12}$$

An intuitive extension of these constraints is the following.

$$\sum_{B \in \mathcal{B}} \theta(B, C)\lambda_B \leq R - 1 \qquad \forall R \geq 2, \forall C \in \mathcal{C}^0_R, \tag{13}$$

However, constraints (13) forbid valid solutions. Indeed, if all pairs (h_r, t_r) in a given $C \in \mathcal{C}^0_R$ belong to the same element B in a partition $\mathcal{P}$, this partition does not induce a cycle of length 0 in the aggregated graph.

To overcome this issue, we define for $C \in V^{2R}$ and $B \in \mathcal{B}$ the value $\phi(B, C) = \min\{1, \theta(B, C)\}$. That is, $\phi(B, C)$ is equal to one if any pair in $\mathcal{F}_C$ is contained in B, and is equal to zero otherwise. This definition allows us to characterize partitions that induce an aggregated graph with cycles of length 0.

Proposition 5. *Given a partition $\mathcal{P} \subseteq \mathcal{B}$, the aggregated graph $G^{\mathcal{P}}$ does not have a cycle of zero length if and only if solution λ induced by $\mathcal{P}$ satisfies the following set of constraints:*

$$\sum_{B \in \mathcal{P}} \phi(B, C)\lambda_B \leq R - 1 \qquad \forall R \geq 2, \forall C \in \mathcal{C}^0_R. \tag{14}$$

Proof. Proof. Assume that there is a cycle of zero length in $G^{\mathcal{P}}$, and take one with the minimum number of nodes, $B_1, \ldots, B_R$. We take for each $r \in \{1, \ldots, R\}$ nodes $h_r, t_r \in B_r$ such that $d(t_r, h_{r+1}) = 0$, where $h_{R+1} = h_1$. Then, for $C = (h_1, \ldots, t_R)$ we have $C \in \mathcal{C}_R^0$ due to the minimality of R. For each $r \in \{1, \ldots, R\}$, we have $\phi(B_r, C) = 1$. Then constraint (14) for this tuple C is violated.

In the other direction, let $C \in \mathcal{C}_R^0$ be a tuple such that the respective constraint in (14) is violated. For each $r \in \{1, \ldots, R\}$ there is at most one set $B \in \mathcal{B}$ such that $\lambda_B = 1$ and $\{h_r, t_r\} \subseteq B$. Then there are R different sets B_r, $r = \{1, \ldots, R\}$, such that $\phi(B_r, C) = 1$ and $\lambda_{B_r} = 1$. By definition of $\mathcal{C}_R^0$, there is a cycle of length 0 which contains nodes $G^{\mathcal{P}}$. □

By Proposition 5, the following IP formulation is valid for the BPPTL$_+^1$.

$$\min \sum_{B \in \mathcal{B}} \lambda_B \tag{15a}$$

$$\text{s.t.} \sum_{B \in \mathcal{B}} \mathbb{1}_B(i) \lambda_B = 1 \qquad \forall i \in V, \tag{15b}$$

$$\sum_{B \in \mathcal{B}} \theta(B, C) \lambda_B \leq R - 1 \qquad \forall R \geq 2, \forall C \in \mathcal{C}_R, \tag{15c}$$

$$\sum_{B \in \mathcal{B}} \phi(B, C) \lambda_B \leq R - 1 \qquad \forall R \geq 2, \forall C \in \mathcal{C}_R^0, \tag{15d}$$

$$\lambda_B \in \{0, 1\} \qquad \forall B \in \mathcal{B}. \tag{15e}$$

Here, constraints (15c) forbid cycles of positive length in the aggregated graph, and constraints (15d) forbid cycles of zero length in the aggregated graph.

4.2 Separation

Given a partition $\mathcal{P} \subseteq \mathcal{B}$ induced by an integer solution, a violated constraint can be separated by finding any cycle in the aggregated graph $G^{\mathcal{P}}$. If the length of the cycle is positive, the corresponding constraint (15c) is violated. If the length of the cycle is zero, the corresponding constraint (15d) is violated. The depth-first-search algorithm can be used to find a cycle in the graph $G^{\mathcal{P}}$ or determine that the graph has no cycle. This can be done in time $\mathcal{O}(|\mathcal{P}| + |A^{\mathcal{P}}|)$.

To separate constraints (15d), we adapt the partial enumeration approach developed in Section 3.2.1. The following proposition is useful for this purpose.

Proposition 6. *Let* $\left(\bar{\lambda}_B\right)_{B \in \mathcal{B}}$ *be a solution satisfying* (15b). *If multigraph* $\mathcal{G} = (V, \mathcal{A})$ *constructed from* $\bar{\lambda}$ *has a cycle* $C = (a_1, a_2, \ldots, a_K)$ *such that* $\sum_{k=1}^K l(a_k) = 0$ *and* $\sum_{k=1}^K v(a_k) < 1$, *and for every* $B \in \mathcal{B}$ *with* $\bar{\lambda}_B > 0$ *there is at most one pair* $\{i_B, j_B\} \in \mathcal{F}_C$ *with* $\{i_B, j_B\} \subseteq B$, *then there is a constraint* (15d) *violated by* $\bar{\lambda}$.

Proof. Proof. Similarly to Proposition 3, we can prove that there is a cycle $C' \subseteq C$ such that constraint $\sum_{B \in \mathcal{B}} \theta(B, C') \lambda_B \leq R - 1$ is violated by $\bar{\lambda}$. For every $B \in \mathcal{B}$ with $\bar{\lambda}_B > 0$ we have $\phi(B, C') = \theta(B, C')$, since at most one of the pairs in $\mathcal{F}_{C'}$ is contained in B. Therefore, constraint (15d) for C' violated by $\bar{\lambda}$. □

The algorithm to search for a violated constraint (15c) or (15d) is similar to Algorithm 1, with two differences. First, when looping over the arcs which leave current node j of current partial path C, we skip arcs $a = (i, j) \in \mathcal{A}_\lambda$ such that there is $B \in \mathcal{B}$ with $\bar{\lambda}_B > 0$ and both $\{i, j\} \subseteq B$ and $\{i', j'\} \subseteq B$ for some $a' = (i', j') \in C \cap \mathcal{A}_\lambda$. Second, in addition to storing cycles with a positive sum of time lags, we also store cycles with the sum of time lags equal to zero.

4.3 Pricing problem

Consider an optimal dual solution $(\bar{\pi}, \bar{\mu}, \bar{\mu}^0)$, with $(\pi_i)_{i \in V}$ of the restricted master problem, where $(\pi_i)_{i \in V}$ are the dual variables of constraints (15b), $(\mu_C)_{C \in \mathcal{C}}$ are the dual variables of constraints (15c), and $(\mu^0_C)_{C \in \mathcal{C}^0}$ are the dual variables of constraints (15d). The binary coefficient of λ_B in the constraint (15d) corresponding to tuple $C \in \mathcal{C}^0$ is equal to $\phi(B, C)$. Recall that $\phi(B, C)$ is equal to one if and only if there is an item pair $\{i, j\} \in \mathcal{F}_C$ such that both i and j are contained in B. Then the reduced cost of variable λ_B is $1 - \sum_{i \in B} \bar{\pi}_i - \sum_{C \in \mathcal{C}} \theta(B, C)\bar{\mu}_C - \sum_{C \in \mathcal{C}^0} \phi(B, C)\bar{\mu}^0_C$.

We now describe a MIP to find a set B of items with the minimum reduced cost. Let $\bar{\mathcal{C}}$ and $\bar{\mathcal{C}^0}$ be the sets of active constraints (15c) and (15d). Variables x and y are defined in the same ways as for formulation (11). Additionally, a binary variable z_C for each cycle $C \in \bar{\mathcal{C}^0}$ determines whether any of the pairs in $\mathcal{F}_C$ is contained in B. Then the pricing problem can be formulated as the following MIP.

$$\max \sum_{i \in V} \bar{\pi}_i x_i + \sum_{\{i,j\} \in \bar{\mathcal{F}}} \left(\sum_{\substack{C \in \bar{\mathcal{C}}: \\ \{i,j\} \in \mathcal{F}_C}} \bar{\mu}_C \right) y_{i,j} + \sum_{C \in \bar{\mathcal{C}^0}} \bar{\mu}^0_C z_C - 1 \tag{16a}$$

$$\text{s.t.} \sum_{i \in V} w_i x_i \leq W, \tag{16b}$$

$$x_i + x_j \leq 1 \qquad \forall \{i, j\} \subseteq V,\ d(i, j) \geq 1 \text{ or } d(j, i) \geq 1, \tag{16c}$$

$$x_i + x_j \leq 1 + y_{i,j} \qquad \forall \{i, j\} \in \bar{\mathcal{F}}, \tag{16d}$$

$$x_i + x_j \leq 1 + z_C \qquad \forall C \in \bar{\mathcal{C}^0}, \forall \{i, j\} \in \mathcal{F}_C, \tag{16e}$$

$$x_i \in \{0, 1\} \qquad \forall i \in V, \tag{16f}$$

$$y_{i,j} \geq 0 \qquad \forall \{i, j\} \in \bar{\mathcal{F}}, \tag{16g}$$

$$z_C \geq 0 \qquad \forall C \in \bar{\mathcal{C}^0}. \tag{16h}$$

Given an optimal solution $(\bar{x}, \bar{y}, \bar{z})$ to (16), we add to the restricted master problem variable $\lambda_{\bar{B}}$, where $\bar{B} = \{i \in V : \bar{x}_i = 1\}$. Since $\mu^0_C < 0$ for $C \in \bar{\mathcal{C}^0}$ and we are maximizing, each variable $\bar{z}_C$ takes the minimum possible value. Therefore, $\bar{z}_C$ is equal to one if and only if $\{i, j\} \subseteq B$ for at least one pair $\{i, j\} \in \mathcal{F}_C$. This implies that z_C is equal to $\phi(B, C)$, and the objective value of solution $(\bar{x}, \bar{y}, \bar{z})$ is equal to the reduced cost of $\lambda_{\bar{B}}$ with the opposite sign.

5 Computational experiments

We implemented the proposed BCP algorithms in C++ language using a generic branch-cut-and-price library BaPCod (Vanderbeck et al., 2020). BapCod uses Cplex 12.8 for solving master and pricing subproblems. We also use Cplex 12.8 to solve compact formulation (2) described in Section 2. The experiments were run on a 2 Deca-core Ivy-Bridge Haswell Intel Xeon E5-2680 v3 server running at 2.50 GHz with 128 GB of RAM. Each instance is solved using a single thread.

5.1 Variant with an unlimited number of available bins

In this section we benchmark different approaches proposed in the paper for the variant of the problem with an unlimited number of available bins. In Section 5.1.1 we describe a new class of instances that we use for benchmarking. In Section 5.1.2 we compare different variants of the compact formulation for the BPPTL. In Section 5.1.3 we test different cut separation algorithms. Finally, in Section 5.1.4 we computationally compare the best variant of the compact formulation and the BCP algorithm.

5.1.1 Generation of instances.

The instances we generate are inspired from the application for the planning of phytosanitary treatments in a vineyard, which is described in Section 1. Consider a set $\mathcal{Q} = \{1, \ldots, Q\}$ of periodic meta-requests for treatment and a planning time horizon $\mathcal{T} = \{1, \ldots, T\}$. For each request $q \in \mathcal{Q}$, the ideal elapsed time between two consecutive treatments is denoted by e_q. For additional flexibility, the minimum and maximum elapsed times between two consecutive treatments of request $q \in \mathcal{Q}$ are set to $e_q^- \leq e_q$ and to $e_q^+ \geq e_q$. Assuming that the planning is embedded in a rolling horizon approach, the last treatment before the planning time horizon and the first treatment after the planning time horizon are fixed. Let $f_q \in \{-e_q + 1, \ldots, 0\}$ be the time period of the last treatment of request $q \in \mathcal{Q}$ before the planning time horizon. Let n_q be the number of treatments of request $q \in \mathcal{Q}$ during the time horizon: it is equal to the maximum integer such that $f_q + n_q e_q \leq T$. Then the time period of the first treatment of request $q \in \mathcal{Q}$ after the planning time horizon is equal to $f'_q = f_q + (n_q + 1) \cdot e_q$. The duration of each treatment i_j^q of request $q \in \mathcal{Q}$ is equal to $w_{i_j^q}$. Treatments are performed by an unlimited fleet of identical vehicles. During a time period, each vehicle is capable of performing treatments of a total duration not exceeding W. The usage cost of a vehicle during one time period is unitary. The objective is to perform all necessary treatments during the planning time horizon while respecting total durations of vehicles and the minimum and maximum elapsed times between any two consecutive treatments of the same request, and to minimize the total vehicle cost.

This problem can be modeled as the BPPTL as follows. Define the bin capacity equal to W and define $L = \infty$. Define two artificial items i^s and i^f with weights $w_{i^s} = w_{i^f} = W$. For each request $q \in \mathcal{Q}$ define n_q items $i_1^q, i_2^q, \ldots, i_{n_q}^q$ with weights equal to $w_{i_1^q}, w_{i_2^q}, \ldots, w_{i_{n_q}^q}$ respectively. In the time-lag graph, we define arc (i^s, i^f) with lag $T+1$, and arc (i^f, i^s) with lag $-(T+1)$. For each request $q \in \mathcal{Q}$ and each treatment $k \in \{1, \ldots, n_q - 1\}$ we define arc (i_k^q, i_{k+1}^q) with lag e_q^-, and arc (i_{k+1}^q, i_k^q) with lag $-e_q^+$. For each $q \in \mathcal{Q}$ we define arc (i^s, i_1^q) with lag $\max\{1, e_q^- + f_q\}$, arc (i_1^q, i^s) with lag $-(e_q^+ + f_q)$, arc $(i_{n_q}^q, i^f)$ with lag $\max\{1, e_a^- - (f'_q - T - 1)\}$, and arc $(i^f, i_{n_q}^q)$ with lag $-(e_a^+ - (f'_q - T - 1))$.

The items created for each request $q \in \mathcal{Q}$ form a double chain in the graph. Each node is connected to the next node in the chain by two arcs, one in each direction. The lags impose the minimum and maximum elapsed time between two consecutive treatments of a request. Items i^s and i^f must be scheduled exactly $T + 1$ time periods apart. Then all other items are scheduled inside T time periods strictly between items i^s and i^f. Thus all treatments are performed within the planning time horizon.

Instances are randomly generated using four integer parameters: the number Q of requests, the number T of time periods, the average number N of request treatments and U the average number of treatments one vehicle can perform in a day. Given tuple (Q, T, N, U), an instance is generated as follows. For each request $q \in \mathcal{Q}$, let e_q be a random integer in the interval $\left[\left\lfloor 0.7 \cdot \frac{T}{N} \right\rfloor, \left\lceil 1.3 \cdot \frac{T}{N} \right\rceil\right]$, and let $e_q^- = \lfloor 0.8 \cdot e_q \rfloor$ and $e_q^+ = \lceil 1.2 \cdot e_q \rceil$. For each request $q \in \mathcal{Q}$, also let f_q be a random integer in interval $[-e_q + 1, 0]$. The bin capacity W is set to 60, and we define the weight $w_{i_j^q}$ for each treatment of request $q \in A$ as a random integer in interval $\left[\left\lfloor 0.4 \cdot \frac{60}{U} \right\rfloor, \left\lceil 1.6 \cdot \frac{60}{U} \right\rceil\right]$. The dataset we generated consists of 252 instances, one instance for each combination of parameters $Q \in \{3, 5, 7, 9\}$, $T \in \{20, 40, 60, 80, 100, 120\}$, $N \in \{4, 7, 10\}$, and $U \in \{2, 3, 4\}$.

In Figure 1 shown in Section 1, we present the time-lag graph for an instance generated with parameters $Q = 7$, $T = 60$, $N = 4$. Parameter U does not have an impact on the time-lag graph. In this instance, random values generated for the ideal elapsed time of a request are $e_1 = 13$, $e_2 = 11$, $e_3 = 20$, $e_4 = 13$, $e_5 = 14$, $e_6 = 20$, and $e_7 = 14$. Random values generated for the last treatment of a request before the planning time horizon are $f_1 = 0$, $f_2 = -8$, $f_3 = -19$, $f_4 = -12$, $f_5 = -5$, $f_6 = -7$, and $f_7 = -12$.

5.1.2 Variants of the compact formulation.

In this section we analyze the performance of some variants of the compact formulation presented in Section 2. Specifically, we seek to determine how different combinations of constraints affect the perfomance of the Cplex solver on our set of instances.

We consider four different variants of formulation (2).

- Capacity constraints. We denote `Cap = Coupled` if we use constraints (2c), and `Cap = Decoupled` if we use constraints (3).
- Time lag constraints. We denote `Pre = Strong` if we use constraints (4) and `Pre = Weak` if we use constraints (2d).
- Breaking symmetries. We denote `Sim1` if we add constraints (5) to the formulation, and `Sim0` otherwise.
- Early and late starts. We denote `Es1` if we apply the early and late start pre-processing, and `Es0` otherwise.

To compare these variants, we used Cplex to solve each of the $2^4 = 16$ combinations, running each instance with a time limit of one hour. The results are presented in Table 1. Columns under "Gap" contain the average optimality gap over all instances, with the gap computed as $\frac{UB-LB}{UB}$. Columns under "# Opt" contain the number of instances solved to optimality. Table 1 shows that the most efficient variant is (`Sim1`, `Es0`, `Cap = Coupled`, `Pre = Weak`). Using this variant of the compact formulation, Cplex solves 98 out of 252 instances.

		Gap				# Opt			
		Sim0		Sim1		Sim0		Sim1	
		Es0	Es1	Es0	Es1	Es0	Es1	Es0	Es1
Cap = Coupled	Pre = Strong	9.9%	9.8%	11.1%	10.7%	80	82	95	96
	Pre = Weak	10.6%	10.3%	11.9%	11.0%	71	72	98	87
Cap = Decoupled	Pre = Strong	32.4%	32.4%	31.3%	31.7%	72	76	88	86
	Pre = Weak	31.5%	25.4%	34.3%	28.0%	77	72	86	89

Table 1: Comparison of different compact formulations.

5.1.3 Comparison of separation approaches.

In Section 3.2.1 two approaches to separate fractional solutions in the BCP algorithm are proposed: solving a MIP and a partial enumeration algorithm. In this section we compare these approaches, as well as the third alternative, which consists in only separating integer solutions. The results are presented in Table 2. Here, column "Gap" presents the average optimality gap, "Root Gap" gives the average gap at the root node of the branch-and-bound tree, "# Opt" the number of instances solved to optimality, and "# Nodes" the average number of explored nodes in the branch-and-bound tree. The results in Table 2 show that the separation of both integer and fractional solutions by enumeration is the approach that allows us to solve to optimality the largest number of instances, and also to get the smallest average optimality gap over all instances. We can also underline that separating fractional solutions is very important for the efficiency of our BCP algorithm.

Separation approach	Gap	Root Gap	# Opt	# Nodes
Only integer solutions	10.6%	12.0%	118	385.7
Fractional solutions by MIP	8.9%	10.5%	148	53.4
Fractional solutions by enumeration	8.7%	10.4%	155	25.6

Table 2: Comparison of different separation methods.

5.1.4 Comparison of the BCP algorithm and Cplex solver.

In this section, we computationally compare two proposed approaches to solve the generated instances with an unlimited number of available bins. Following the results in Section 5.1.2, as the first approach we use the variant (`Sim1`, `Es0`, `Cap = Coupled`, `Pre = Weak`) of the compact formulation solved by Cplex. The second approach is our BCP algorithm with the enumeration separation algorithm applied to fractional solutions due to the results we obtained in Section 5.1.3. Both approaches are run with the time limit of one hour.

Table 3 shows the results of this computational comparison. We give results for different subsets of instances grouped by the generation parameters used, as well as the overall results. In the table, column "# Inst." shows the number of instances in the respective subset. Columns under "% Opt" show the percentage of the instances that were solved to optimality within the time limit. Columns under "Gap" show the average optimality gap, columns under "Time" show the average time in seconds. Column "Root Gap" shows the average optimality gap at the root node of the branch-and-bound tree in BCP algorithm. This gap is computed as $\frac{UB-LB}{UB}$, where LB in this case is the lower bound obtained by the column and cut generation in the root node, and UB is the value of the best solution obtained within the time limit. Finally, column "# Nodes" shows the average number of nodes explored in the branch-and-bound tree of the BCP algorithm. The detailed instance-by-instance results are available in the online supplement.

	# Inst.	% Opt		Gap		Time		Root Gap	# Nodes
		BCP	Cplex	BCP	Cplex	BCP	Cplex	BCP	BCP
$T = 20$	36	55.6%	55.6%	3.0%	2.9%	1720.2	1815.1	3.9	56.4
$T = 40$	36	58.3%	47.2%	9.1%	8.8%	1655.8	2179.1	10.4	19.7
$T = 60$	36	63.9%	47.2%	11.0%	9.6%	1511.0	2266.8	12.5	23.8
$T = 80$	36	61.1%	36.1%	7.7%	12.1%	1579.0	2447.5	9.0	18.8
$T = 100$	36	63.9%	30.6%	11.8%	15.7%	1549.7	2751.9	13.9	23.8
$T = 120$	36	63.9%	30.6%	8.9%	16.3%	1403.6	2763.3	10.9	23.9
$T = 140$	36	63.9%	25.0%	9.4%	18.1%	1388.8	2937.3	11.9	12.7
$Q = 3$	63	96.8%	66.7%	0.2%	5.1%	130.0	1477.9	1.6	7.3
$Q = 5$	63	68.3%	38.1%	3.4%	10.8%	1480.1	2406.0	6.7	42.7
$Q = 7$	63	44.4%	33.3%	7.7%	13.0%	2129.0	2689.5	8.8	27.0
$Q = 9$	63	36.5%	17.5%	23.5%	18.8%	2437.0	3232.9	24.4	25.3
$N = 4$	84	97.6%	79.8%	0.2%	1.3%	201.5	1083.5	2.1	6.0
$N = 7$	84	57.1%	34.5%	5.6%	10.4%	1802.2	2687.5	7.2	35.8
$N = 10$	84	29.8%	2.4%	20.3%	24.1%	2628.2	3583.7	21.8	35.0
$U = 2$	84	77.4%	31.0%	1.3%	10.6%	991.1	2613.3	3.3	51.7
$U = 3$	84	56.0%	38.1%	10.1%	11.3%	1744.5	2436.1	11.3	13.7
$U = 4$	84	51.2%	47.6%	14.7%	13.8%	1896.4	2305.3	16.5	11.4
All	252	61.5%	38.9%	8.7%	11.9%	1544.0	2451.6	10.4	25.6

Table 3: Results of the computational comparison between the best variant of the compact formulation solved by Cplex and the best variant of the BCP algorithm

The results, presented in Table 3 show that our BCP algorithm is substantially more efficient than the Cplex solver applied to the compact formulation of the problem. When considering the results for different subsets of instances, we note that the BCP algorithm is not very "sensitive" to the length of the time horizon, whereas the efficiency of Cplex decreases with the increase of parameter T. This is not surprising, as the size of the compact formulation depends heavily on T, whereas the theoretical and practical computational complexity of the components of the BCP algorithm never depends on T. On the other hand, when parameters Q or N increase, both the Cplex and the BCP algorithm become less efficient. The impact of parameter U is however very different. Smaller values of U are better for the BCP algorithm, and larger values are better for Cplex. This behaviour is similar to that observed when solving the classical BPP. When the average number of items which can fit into one bin increases, the quality of lower bounds obtained by the linear relaxation of compact formulations converges to the quality of the column generation lower bound. As the quality of lower bounds become similar, MIP solvers start to be more efficient than column generation approaches due to the fact the linear relaxation of compact formulations can be solved much faster.

5.2 Variant with one available bin per time period

In this section we computationally estimate the efficiency of our BCP algorithm for the variant of the BPPTL with one available bin per time period, i.e. for the BPPTL^1_+. We also compare the BCP algorithm to other approaches available in the literature for special cases of the BPPTL^1_+.

5.2.1 Instances.

We tested the branch-cut-and-price algorithm on the BPP-GP instances considered by Kramer et al. (2017). They created a set of instances for the BPP-GP by extending a benchmark set for the SALBP-1 proposed in (Otto et al., 2013). For each instance of the SALBP-1, they created two instances of the BPP-GP by defining random time lags on the arcs of the precedence graph, one of them with random time lag values in $\{0, 1\}$ and the other with random time lag values in $\{0, 1, 2, 3\}$. We will refer to these two special cases of the BPP-GP as BPP-GP01 and BPP-GP03. The literature instances for the BPP-P and the SALBP-1 were also considered by Kramer et al. (2017), as these two problems are special cases of BPP-GP01.

The authors of (Kramer et al., 2017) kindly provided us with the results of their algorithm for the instances of the SALBP-1, the BPP-P, the BPP-GP01, and the BPP-GP03. As mentioned in the introduction, the algorithm in (Kramer et al., 2017) is designed for the makespan objective function. Thus, the problem considered in (Kramer et al., 2017) is a special case of the BPPTL^1_+ only if all time lag values are in $\{0, 1\}$. Thus, we tested our BCP algorithm only on the instances of the SALBP-1, the BPP-P, and the BPP-GP01. Instances containing 20, 50, and 100 items are used. We skip instances with 1000 items as they are out of reach for our algorithm. For each problem variant and each value n equal to the number of items, there are 525 instances in the test set. The structure of precedence graphs is described in (Otto et al., 2013).

Kramer et al. (2017) propose a heuristic procedure to find feasible solutions and thus upper bounds. They also use known techniques from the literature to compute lower bounds. The best upper and lower bounds reported in (Kramer et al., 2017) are usually very good. Most of the instances are solved to optimality. A summary of known results can be seen in Table 4. Column "# Not Opt" shows the number of instances for which the best known lower bound is strictly smaller than the best known upper bound. Column "# Opt" gives the number of instances for which the best known bounds are equal, i.e. the number of instances with known optimum solutions. In column "Gap Unsolved", the average optimality gap is shown for the open instances.

# Items	type	# Not Opt	# Opt	Gap Unsolved
$n = 20$	BPP-GP01	0	525	-
	BPP-P	0	525	-
	SALBP-1	0	525	-
$n = 50$	BPP-GP01	15	510	3.9%
	BPP-P	26	499	3.8%
	SALBP-1	2	523	8.7%
$n = 100$	BPP-GP01	67	458	2.8%
	BPP-P	34	491	3.1%
	SALBP-1	26	499	2.7%

Table 4: Number of instances solved to optimality in (Kramer et al., 2017) and average optimality gap for unsolved instances

5.2.2 Experimental results of the BCP algorithm.

In this section, we test our BCP algorithm on the instances described in Section 5.2.1. We use two variants of the BCP algorithm. First variant BCP-K is initialized with the best known upper bounds obtained in (Kramer et al., 2017). Second variant BCP-∞ does not uses any initial upper bounds. We impose the time limit of one hour for each instance. Given an instance, we denote by LB(ALG) and UB(ALG) the best lower and upper bounds obtained by algorithm ALG, which can be either BCP-K, BCP-∞, or KDI, where the latter is the approach proposed in (Kramer et al., 2017).

In Table 5 we compare upper bounds UB(BCP-∞) and UB(KDI). Instances are divided into three groups: "Better", "Equal" and "Worse", depending on whether UB(KDI)$>$ UB(BCP-∞), UB(KDI)$=$ UB(BCP-∞), and UB(KDI)$<$ UB(BCP-∞). Columns under "Avg. Diff." give the average absolute difference between values UB(KDI) and UB(BCP-∞) for the corresponding group of instances. For some groups of instances, this average difference is marked with "∞", which means that for at least one instance in this group no feasible solution was found by algorithm BCP-∞.

		Better		Equal	Worse	
# Items	type	#	Avg. Diff.	#	#	Avg. Diff.
$n = 20$	BPP-GP01	0	-	525	0	-
	BPP-P	0	-	525	0	-
	SALBP-1	0	-	525	0	-
$n = 50$	BPP-GP01	1	1.0	505	19	1.0
	BPP-P	0	-	467	58	1.2
	SALBP-1	0	-	517	8	1.0
$n = 100$	BPP-GP01	7	1.0	305	213	∞
	BPP-P	9	1.0	266	250	∞
	SALBP-1	11	1.0	325	189	∞

Table 5: Comparison of upper bounds UB(BCP-∞) and UB(KDI)

The results in Table 5 show that the heuristic proposed in (Kramer et al., 2017) is usually much better to obtain good feasible solutions than the algorithm BCP-∞. Nevertheless, our algorithm is able to improve the best known solutions for 28 instances.

In the following experiments we use the variant BCP-K of our algorithm. This implies that the best known upper bound found by the BCP algorithm cannot be worse than UB(KDI). Table 6 shows the results of the comparison of upper bounds UB(BCP-K) and UB(KDI). Our

algorithm BCP-K is able to improve the best known solutions for three more instances. For a large majority of instances, our algorithm is not able to improve the best known upper bound. This is not surprising, as most of best known upper bounds are optimal values.

		Better		Equal
# Items	type	#	Avg. Diff.	#
$n = 20$	BPP-GP01	0	-	525
	BPP-P	0	-	525
	SALBP-1	0	-	525
$n = 50$	BPP-GP01	1	1.0	524
	BPP-P	0	-	525
	SALBP-1	0	-	525
$n = 100$	BPP-GP01	9	1.0	516
	BPP-P	10	1.0	515
	SALBP-1	11	1.0	514

Table 6: Comparison of upper bounds UB(BCP-K) and UB(KDI)

In Table 7, we compare lower bounds LB(BCP-K) and LB(KDI). In the same vein as above, the instances are divided into the groups "Better", "Equal" and "Worse", depending on whether LB(KDI)< LB(BCP-K), LB(KDI)= LB(BCP-K), and LB(KDI)> LB(BCP-K). The results show that the lower bounds obtained by our algorithm BCP-K are on average worse than the best known ones for instances of the BPP-P and the SALBP-1. However, our lower bounds are on average better than the best known ones for instances of the BPP-GP01. This is not surprising as the BPP-GP has been less studied in the literature. Overall, our algorithm improved 88 best known lower bounds among 170 open instances, i.e. for the majority of open instances.

		Better		Equal	Worse	
# Items	type	#	Avg. Diff.	#	#	Avg. Diff.
$n = 20$	BPP-GP01	0	-	525	0	-
	BPP-P	0	-	525	0	-
	SALBP-1	0	-	525	0	-
$n = 50$	BPP-GP01	13	1.0	508	4	1.0
	BPP-P	25	1.0	467	33	1.1
	SALBP-1	1	1.0	522	2	1.0
$n = 100$	BPP-GP01	35	1.4	456	34	1.1
	BPP-P	8	1.0	424	93	1.2
	SALBP-1	6	1.0	489	30	1.0

Table 7: Comparison of lower bounds LB(BCP-K) and LB(KDI)

In Table 8, we show the optimality status of instances before and after applying our algorithm. Column "# Opt KDI" shows the number of instances with known optimum solutions in the literature. Column "# Opt. +BCP" gives the number of instances with known optimality based on the literature results and the results obtained by our BCP algorithm. Finally, "# Not Opt." shows the number of instances that still remain open. This table shows that for 119 instances the optimality status is obtained for the first time among 170 open instances. 51 instance still remain open.

Finally, in Table 9, we give statistics for our algorithm BCP-K applied to instances described in Section 5.2.1. The detailed instance-by-instance results are available in the online supplement.

# Items	type	# Opt KDI	# Opt. +BCP	# Not Opt.
$n = 20$	BPP-GP01	525	0	0
	BPP-P	525	0	0
	SALBP-1	525	0	0
$n = 50$	BPP-GP01	510	14	1
	BPP-P	499	25	1
	SALBP-1	523	1	1
$n = 100$	BPP-GP01	458	44	23
	BPP-P	491	18	16
	SALBP-1	499	17	9

Table 8: Optimality status of instances

Columns under "# Nodes" show the average number of explored nodes in the branch-and-bound tree for both unsolved instances and instances solved to optimality. Column "Not Solved" gives the number of instances not solved to optimality by the algorithm BCP-K. Column "Solved" shows the number of instances solved to optimality. Column "Time" gives the average time in seconds our algorithm took when it could solve instances to optimality. It can be seen from the results in Table 9 that algorithm BCP-K is less efficient than the approach by Kramer et al. (2017) for the BPP-P and the SALBP-1, but more efficient for the BPP-GP01. We should mention, however, that our algorithm is initialized by the best known solutions obtained in (Kramer et al., 2017). Thus, it makes sense to combine our BCP algorithm with the approach proposed in (Kramer et al., 2017) to solve instances of the BPP-GP01. These two methods seem to have a complementary strength, according to the results presented in Table 8.

		# Nodes		# Instances		Time
# Items	type	Not Solved	Solved	Not Solved	Solved	Solved
$n = 20$	BPP-GP01	-	1.3	0	525	0.47
	BPP-P	-	2.0	0	525	0.57
	SALBP-1	-	1.1	0	525	0.70
$n = 50$	BPP-GP01	1537.4	7.6	5	520	35.09
	BPP-P	820.8	28.0	34	491	82.89
	SALBP-1	89.7	4.6	3	522	12.52
$n = 100$	BPP-GP01	246.6	4.1	55	470	81.04
	BPP-P	288.1	22.4	108	417	149.73
	SALBP-1	134.6	5.4	39	486	124.57

Table 9: Statistics of the BCP-K algorithm

6 Conclusions

In this work, we introduce the bin packing problem with time lags (BPPTL). As a motivation, we describe an application of the BPPTL in which one needs to decide on the planning of phytosanitary treatments in a vineyard. We first present an IP formulation for the problem. Then for two important special cases of the BPPTL with an unlimited number of available bins and with one available bin per time period, we propose an exact branch-cut-and-price (BCP) algorithm. This algorithm relies on a known IP formulation with an exponential number of variables which define a relaxation for the problem, and a new family of constraints which serve to cut infeasible solutions produced by the relaxation. We present two approaches to separate

the new family of constraints. The BCP algorithm also incorporates an automatic dual price smoothing technique to improve the convergence of column generation, a strong diving heuristic for finding feasible solutions, and a strong multi-phase Ryan&Foster branching.

The computational experiments show that our BCP algorithm substantially outperforms a commercial MIP solver on newly generated instances inspired from the phytosanitary treatment planning application. The experiments on literature instances for special cases of the BPPTL show that the BCP algorithm can complement known approaches, as it allows us to obtain the optimality status for 70% of previously open instances.

Nevertheless, we believe that the efficiency of our BCP algorithm can still be improved. When solving larger instances, the bottleneck of the algorithm is solving the pricing problem, which is the knapsack problem with hard and soft conflicts (KPHSC). Approaching the KPHSC directly by applying a MIP solver may not be the best way to solve it, especially when a bin can hold few items on average. The KPHSC itself is a new and very interesting problem which would be interesting to study in the future. There are efficient exact algorithms in the literature for the related knapsack problem with conflicts (KPC), such as the ones proposed in (Bettinelli et al., 2017; Coniglio et al., 2020; Wei et al., 2020). These algorithms should be adapted or new algorithms should be developed for the KPHSC.

The experiments on literature instances show that another component of the BCP algorithm which should be improved is the primal heuristic. The BCP algorithm currently relies on the generic strong diving heuristic, which can be slow and relatively inefficient for larger instances. Therefore, developing specialized heuristics for the BPPTL is an important research direction. Known heuristics for the related problem BPP-GP, such as the one proposed in (Kramer et al., 2017), may serve as a basis for this work.

The BCP algorithm is currently limited to two special cases of the BPPTL. Extending it to the general case is an interesting research perspective. This seems not to be easy to do, as in general it is an NP-complete problem (Finta and Liu, 1996) to determine whether a partition of items into bins is feasible or not with respect to the time lags. Thus, any family of cutting planes to cut off infeasible assignments of items to bins will be NP-hard to separate even if only integer assignments are considered. An extension to the general case is relevant in practice, as sometimes we cannot assume that we may use any number of resources as we want in any time period.

In the phytosanitary treatment planning application, we assume that the time needed for a vehicle to go from one sector to another is negligible in comparison to the duration of treatments. Thus transition times of vehicles are ignored. In practice, this is not always true. If transition times are not negligible, the problem shifts from the class of precedence-constrained bin packing problems to the class of periodic vehicle routing problems (Campbell and Wilson, 2014).

In the standard periodic vehicle routing problem (PVRP), there are only a few alternatives for visiting each client. For example, a client may be visited on Monday and Thursday, or on Tuesday and Friday. If the planning time horizon is sufficiently long, having only a few alternatives dramatically limits the flexibility of visits. Flexible variants of the PVRP exist, such as one proposed in (Archetti et al., 2017). However, flexibility in that case concerns the amount of goods delivered to a client during each visit. An extension of the BPPTL to a flexible periodic vehicle routing problem in which one imposes minimum and maximum time lags on consecutive visits to the same client has a large number of applications in the context of provision of services such as periodic maintenance visits. A BCP algorithm similar to the one proposed in this paper may be efficient for such a flexible periodic vehicle routing problem, provided that the pricing problem takes into account the vehicle transition times.

Finally, the BPPTL structure is sufficiently rich that there may be other applications which result in time lag graphs in a class different from the class of double-linked chain graphs. Finding such applications and suggesting test instances for them is useful for further progress in solution approaches for the BPPTL.

Acknowledgments

We thank Raphael Kramer for kindly providing us with results for individual instances and other details on their work.

Experiments presented in this paper were carried out using the PlaFRIM (Federative Platform for Research in Computer Science and Mathematics), created under the Inria PlaFRIM development action with support from Bordeaux INP, LABRI and IMB and other entities: Conseil Régional d'Aquitaine, Université de Bordeaux, CNRS and ANR in accordance to the "Programme d'Investissements d'Avenir" (see `www.plafrim.fr/en/home`).

This work was supported by the Government of Chile through CONICYT-PFCHA/Doctorado Nacional/2017-211713157 (ORL).

References

Tobias Achterberg. *Constraint Integer Programming*. PhD thesis, Technische Universitat Berlin, 2007.

Claudia Archetti, Elena Fernández, and Diana L. Huerta-Muñoz. The flexible periodic vehicle routing problem. *Computers & Operations Research*, 85:58 – 70, 2017.

C. Artigues. On the strength of time-indexed formulations for the resource-constrained project scheduling problem. *Operations Research Letters*, 45(2):154–159, 2017.

M Bartusch, R H Möhring, and F J Radermacher. Scheduling project networks with resource constraints and time windows. *Annals of Operations Research*, 16(1):199–240, dec 1988. ISSN 1572-9338. doi: 10.1007/BF02283745. URL `https://doi.org/10.1007/BF02283745`.

Andrea Bettinelli, Valentina Cacchiani, and Enrico Malaguti. A branch-and-bound algorithm for the knapsack problem with conflict graph. *INFORMS Journal on Computing*, 29(3):457–473, 2017.

Ann Melissa Campbell and Jill Hardin Wilson. Forty years of periodic vehicle routing. *Networks*, 63(1):2–15, 2014.

N. Christofides, R. Alvarez-Valdés, and J.M. Tamarit. Project scheduling with resource constraints: A branch and bound approach. *European Journal of Operational Research*, 29(3): 262–273, 1987.

Stefano Coniglio, Fabio Furini, and Pablo San Segundo. A new combinatorial branch-and-bound algorithm for the knapsack problem with conflicts. *European Journal of Operational Research*, online, 2020.

Mauro Dell'Amico, José Carlos Díaz Díaz, and Manuel Iori. The Bin Packing Problem with Precedence Constraints. *Operations Research*, 60(6):1491–1504, 2012. doi: 10.1287/opre.1120.1109. URL `https://doi.org/10.1287/opre.1120.1109`.

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

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

Samir Elhedhli, Lingzi Li, Mariem Gzara, and Joe Naoum-Sawaya. A Branch-and-Price Algorithm for the Bin Packing Problem with Conflicts. *INFORMS Journal on Computing*, 23(3): 404–415, 2011.

Albert E. Fernandes Muritiba, Manuel Iori, Enrico Malaguti, and Paolo Toth. Algorithms for the Bin Packing Problem with Conflicts. *INFORMS Journal on Computing*, 22:401–415, 2010.

Lucian Finta and Zhen Liu. Single machine scheduling subject to precedence delays. *Discrete Applied Mathematics*, 70(3):247–266, 1996. ISSN 0166-218X. doi: https://doi.org/10.1016/0166-218X(96)00110-2. URL http://www.sciencedirect.com/science/article/pii/0166218X96001102.

Michel Gendreau, Gilbert Laporte, and Frédéric Semet. Heuristics and lower bounds for the bin packing problem with conflicts. *Computers and Operations Research*, 31(3):347 – 358, 2004.

C Joncour, S Michel, R Sadykov, D Sverdlov, and F Vanderbeck. Column Generation based Primal Heuristics. *Electronic Notes in Discrete Mathematics*, 36:695–702, 2010. ISSN 1571-0653. doi: https://doi.org/10.1016/j.endm.2010.05.088. URL http://www.sciencedirect.com/science/article/pii/S1571065310000892.

Leonid V Kantorovich. Mathematical methods of organizing and planning production. *Management science*, 6(4):366–422, 1960. Translation of a 1939 paper in Russian.

Ali Khanafer, François Clautiaux, and El-Ghazali Talbi. New lower bounds for bin packing problems with conflicts. *European Journal of Operational Research*, 206(2):281 – 288, 2010.

Ali Khanafer, François Clautiaux, and El-Ghazali Talbi. Tree-decomposition based heuristics for the two-dimensional bin packing problem with conflicts. *Computers & Operations Research*, 39(1):54 – 63, 2012.

Raphael Kramer, Mauro Dell'Amico, and Manuel Iori. A batching-move iterated local search algorithm for the bin packing problem with generalized precedence constraints. *International Journal of Production Research*, 55(21):6288–6304, 2017. doi: 10.1080/00207543.2017.1341065. URL https://doi.org/10.1080/00207543.2017.1341065.

O Kullmann. *Handbook of Satisfiability*, chapter Fundaments of branching heuristics, pages 205–244. IOS Press, Amsterdam, 2009.

David R Morrison, Edward C Sewell, and Sheldon H Jacobson. An application of the branch, bound, and remember algorithm to a new simple assembly line balancing dataset. *European Journal of Operational Research*, 236(2):403–409, 2014. ISSN 0377-2217. doi: https://doi.org/10.1016/j.ejor.2013.11.033. URL http://www.sciencedirect.com/science/article/pii/S0377221713009508.

Alena Otto, Christian Otto, and Armin Scholl. Systematic data generation and test design for solution algorithms on the example of SALBPGen for assembly line balancing. *European Journal of Operational Research*, 228(1):33–45, 2013. ISSN 0377-2217. doi: https://doi.org/10.1016/j.ejor.2012.12.029. URL http://www.sciencedirect.com/science/article/pii/S0377221713000039.

Jordi Pereira. Empirical evaluation of lower bounding methods for the simple assembly line balancing problem. *International Journal of Production Research*, 53(11):3327–3340, 2015. doi: 10.1080/00207543.2014.980014. URL https://doi.org/10.1080/00207543.2014.980014.

Jordi Pereira. Procedures for the bin packing problem with precedence constraints. *European Journal of Operational Research*, 250(3):794–806, 2016. ISSN 0377-2217. doi: https://doi.org/10.1016/j.ejor.2015.10.048. URL http://www.sciencedirect.com/science/article/pii/S0377221715009741.

Artur Pessoa, Ruslan Sadykov, Eduardo Uchoa, and François Vanderbeck. Automation and combination of linear-programming based stabilization techniques in column generation. *INFORMS Journal on Computing*, 30(2):339–360, 2018.

Artur Pessoa, Ruslan Sadykov, Eduardo Uchoa, and François Vanderbeck. A generic exact solver for vehicle routing and related problems. *Mathematical Programming B*, 183:483–523, 2020.

D. M. Ryan and B. A. Foster. An integer programming approach to scheduling. In A. Wren, editor, *Computer Scheduling of Public Transport Urban Passenger Vehicle and Crew Scheduling*, pages 269 – 280. North-Holland, Amsterdam, 1981.

Ruslan Sadykov and François Vanderbeck. Bin Packing with Conflicts: A Generic Branch-and-Price Algorithm. *INFORMS Journal on Computing*, 25(2):244–255, 2013. doi: 10.1287/ijoc.1120.0499. URL http://dx.doi.org/10.1287/ijoc.1120.0499.

Ruslan Sadykov, François Vanderbeck, Artur Pessoa, Issam Tahiri, and Eduardo Uchoa. Primal Heuristics for Branch and Price: The Assets of Diving Methods. *INFORMS Journal on Computing*, 31(2):251–267, 2019. doi: 10.1287/ijoc.2018.0822. URL https://doi.org/10.1287/ijoc.2018.0822.

Armin Scholl and Christian Becker. State-of-the-art exact and heuristic solution procedures for simple assembly line balancing. *European Journal of Operational Research*, 168(3):666–693, 2006. ISSN 0377-2217. doi: https://doi.org/10.1016/j.ejor.2004.07.022. URL http://www.sciencedirect.com/science/article/pii/S0377221704004795.

François Vanderbeck, Ruslan Sadykov, and Issam Tahiri. BaPCod — a generic Branch-And-Price Code. https://realopt.bordeaux.inria.fr/?page_id=2, 2020.

Laguna Wei, Zhixing Luo, Roberto Baldacci, and Andrew Lim. A new branch-and-price-and-cut algorithm for one-dimensional bin-packing problems. *INFORMS Journal on Computing*, 32 (2):428–443, 2020.