CHALMERS
UNIVERSITY OF TECHNOLOGY

On the Use of Equivalence Classes for Optimal and Suboptimal Bin Packing and Bin Covering

On the Use of Equivalence Classes for Optimal and Sub-Optimal Bin Packing and Bin Covering

On the Use of Equivalence Classes for Optimal and Sub-Optimal Bin Packing and Bin Covering

Sabino Roselli[1], Fredrik Hagebring[1], Sarmad Riazi[1], Martin Fabian[1], Knut Åkesson[1]

***Abstract*—*Bin packing* and *bin covering* are important optimization problems in many industrial fields, such as packaging, recycling, and food processing. The problem concerns a set of *items*, each with its own *value*, that are to be sorted into *bins* in such a way that the total value of each bin, as measured by the sum of its item values, is not above (for packing) or below (for covering) a given *target* value. The optimization problem concerns minimizing, for bin packing, or maximizing, for bin covering, the number of bins. This is a combinatorial NP-hard problem, for which true optimal solutions can only be calculated in specific cases, such as when restricted to a small number of items. To get around this problem, many sub-optimal approaches exist. This paper describes formulations of the bin packing and covering problems that allows to find the true optimum for instances counting hundreds of items using general purpose MILP-solvers. Also presented are sub-optimal solutions that come within less than 10% of the optimum, while taking significantly less time to calculate, even ten to 100 times faster, depending on the required accuracy.**

***Note to Practitioners*—A typical case for bin covering is in food processing where food items are automatically sorted into trays of similar weight, so that the overweight is minimized. Another application is in recycling, where items like batteries should be put in crates of similar weight, so that the crates do not exceed a target weight, due to later manual handling, but at the same time we want as few crates as possible. This is a bin packing problem. On an industrial scale these tasks are fully automated. Though modern software tool's efficiency to solve bin sorting problems have increased significantly in later years, the problems are inherently tough in the sense that the solution time grows exponentially with the number of items. This limits the problem sizes that can be solved to optimality within reasonable time. Therefore, much research has focused on heuristic rules that give reasonable solving times while not giving the true optimal number of bins. However, in many cases the true optimal solution is preferable, and sometimes even necessary, so this is an industrially interesting problem. This paper describes an approach to solve the bin packing and covering problems to the true optimum that increases the limit of the number of items that can typically be handled. This is done by observing that items of same value need not be distinguished. Instead, we can formulate packing/covering problems over item *values* rather than individual items, and sort integer numbers of these values into bins, which allows to solve to optimum for more than 500 items in reasonable time. In addition, by redefining what we mean by *same value*, we can consider more items to have the same value and achieve even better calculational efficiency.**

Work supported by VR SyTeC (2016-06204), and the Chalmers Production Area of Advance. Department of Electrical Engineering Chalmers University of Technology Göteborg, Sweden. Part of this work was presented in CASE 2019. [1]{rsabino, fredrik.hagebring, sarmad.riazi, fabian, knut}@chalmers.se.

I. Introduction

The (one-dimensional) *bin sorting* problem concerns sorting items with given values into *bins* such that the value of a bin, counted as the sum of its included values, conforms to a specified *target* value, while at the same time optimizing the total number of bins. Two (dual) variants of this problem exist, *bin packing* [1] where the bin values cannot go over, and *bin covering* [2] where the bin values cannot go under, respectively, the target value. The problems are NP-hard [1] combinatorial optimization problems, meaning that there is no general algorithm that can solve either problem for an arbitrary number of items in reasonable time.

Due to this, different heuristic algorithms to provide sub-optimal solutions while guaranteeing a bound from the optimum and having polynomial complexity have been a long-time active field of research. Recent surveys of related problems are presented in [1] for approximative algorithms and in [3] for exact algorithms.

Recent work on bin sorting is based on branch-and-price based algorithms, see e.g. [4]. Pseudo-polynomial formulations, [5], [6], [7], allow a more compact formulation and avoid the complexity introduced in the implementation of branch-and-price based algorithms. Of interest in some practical applications is also temporal extensions, [8], where the capacity of a bin must be consumed within a given time window.

In our previous work on the subject [9], we presented a model to solve the bin covering problem to optimality by means of *Mixed Integer Linear Programming* (MILP), and we showed through a computational analysis of more than 800 problem instances that a MILP solver is able to handle quickly large sized instances. In this work we extend the analysis to include also bin packing, and we provide mathematical proof of the validity of our claims on top of which we implemented the above-mentioned formulation. We also present a sub-optimal approach based on a simplification of the original instances that exploits the feature of our new formulation and guarantees close to optimal results.

In industrial problems there are often additional constraints that are not included in text book formulations of the sorting problems. An example from an industrial application: for bin covering problems it might be allowed to go a few percent below the target weight for certain bins as long as the average bin value is on or above target. Algorithms that are tailor made for textbook formulations of the sorting problems might have problems to generalize to such modified versions of the problem. In industrial applications general purpose solvers are thus often preferred due to their ability generalize to new problem formulations. So, though we in this paper treat only the text book versions of bin sorting, we do so by focusing on formulations that

can be given as input to general-purpose MILP solvers, and we evaluate the efficiency of these formulations.

First is presented the standard formulation that can be found in most text books. This formulation was first introduced by [10] and is typically useful only for a small number of items. Then is given the "subset" formulation, which removes the identities of the bins and thus allows to solve for a much larger number of items and bins. Thirdly is given the "equivalence class" formulation, which further removes the identities between items of same values, and hence allows to solve for even larger numbers of items and bins. As explained later on in the paper, the idea of *equivalence classes* leads to define combinations of items that meet the target requirement (i.e. the cumulative value of such items is below the target for the bin packing and above it for the bin covering); such combinations are given the name of *packages* in this paper and, to the best of our knowledge, they have first been introduced by [11] to implement a specific purpose algorithm to solve bin covering problem, and then used in [12] to implement a branch and price algorithm, while we use it to develop a linear-integer model for a general purpose solver. Also, we improve the concept by defining a subset of packages among which bins can be selected that still leads to the optimal solution. In other words, we do not need to enumerate all possible combinations of items in order to find the optimal solution, but only a rather small portion of it.

The contributions to this paper are: (i) improving the concepts of equivalence classes by introducing the notion of *skinny* and *fit* bins for the bin covering and the bin packing respectively; proving that the the optimal solution of an instance of the bin packing (covering) is only composed by *fit* (*skinny*) bins; (iii) develop a heuristic method for both the bin covering and bin packing problem, based on equivalence classes, that significantly reduces the computation time while still guaranteeing close to optimum results; (iiii) evaluating the method against other algorithms for bin sorting problems by running it over different sets of benchmark instances.

In the next section the general bin sorting problem is described, giving three different MILP formulations, two of which exploit the fact that prospective bins can be pre-calculated to make the MILP solver's job easier. Then Section III presents how the combinatorial explosion can be further mitigated by restricting the number of pre-calculated bins, while still guaranteeing optimal solutions. Section IV then describes how the number of pre-calculated bins can be made even smaller, but then not guaranteeing optimal solutions. The experimental results of Section V shows the computational benefits of both the optimal formulations, and the sub-optimal formulation that comes within less than 10% of the optimum, while achieving a significant reduction in computation time. The paper is concluded in Section VI.

II. Bin Sorting

Bin sorting is a generic term for the two (dual) problems of bin *packing* and bin *covering*. The bin sorting problem concerns a set of *items* $V = \{v_1, v_2, \ldots, v_n\}$, each with a *value* so that there can be defined an ordering between the items, such that $v_1 \geq v_2 \geq \cdots \geq v_n$. For notational simplicity, except for in a few places, the distinction between an item and its value will not be made; note though that items are unique, whereas two items can have the same values. Given a subset $V' \subseteq V$ we denote its minimum and maximum values as $\min(V')$ and $\max(V')$, respectively.

A *bin* $b_j \subseteq V$ is a subset of V. For a bin b_j we can define its value B_j as the sum of the item values it contains, that is, $B_j = \sum_{v_i \in b_j} v_i$.

The bin sorting problem can now be defined as a tuple $\langle V, t, \bowtie \rangle$, where t is a target value that defines a bound on the bin values, and $\bowtie$ is $\leq$ for bin packing, and $\geq$ for bin covering. The problem is now to find an optimal solution $B_{opt} = \{b_1, b_2, \ldots, b_m\}$, which is a partition of V with the minimum, for packing, or maximum, for covering, number m of bins, such that $\forall b_j \in B_{opt},\ B_j \bowtie t$. It is assumed that $\sum_{v_i \in V} v_i > t$, and $\forall v_i \in V\ v_i < t$.

Since we in large parts of the paper simultaneously deal with both covering and packing, we have introduced a non-standard notation of our own (such as "target" t instead of "capacity" c). This so, since the communities dealing with the respective problems do not always agree on the notation. Furthermore, the term "bin covering" is sometimes used to denote a different problem, where the number of bins is fixed and the problem is maximizing the number of packed items while not exceeding the target value for any bin [13].

A. The Standard Formulation

One way to formulate the bin sorting problem is as a *mixed linear integer programming* (MILP) problem, where the decision variables represent bins and the allocation of items to the bins. Let $b_j\ (j = 1, \ldots, n)$ be 0-1-variables representing whether a certain bin is used ($b_j = 1$) or not ($b_j = 0$), and let $x_{ij}\ (i, j = 1, \ldots, n)$ be 0-1-variables representing whether the value v_i is assigned to the j'th bin ($x_{ij} = 1$), or not ($x_{ij} = 0$). The MILP problem can then be formulated as:

$$\min / \max \sum_{j=1}^{n} b_j \text{ subject to} \tag{1}$$

$$\sum_{j=1}^{n} x_{ij} = 1 \qquad \forall i = 1, \ldots, n \tag{2}$$

$$\sum_{i=1}^{n} x_{ij} \cdot v_i \bowtie b_j \cdot t \qquad \forall j = 1, \ldots, n \tag{3}$$

$$x_{ij} \in \{0, 1\} \qquad \forall i, j = 1, \ldots, n \tag{4}$$

$$b_j \in \{0, 1\} \qquad \forall j = 1, \ldots, n \tag{5}$$

The objective function (1) is the sum over all the variables representing whether a certain bin is used or not, and since these are binary 0-1-variables, the sum is the number of used bins; this sum is to be minimized for bin packing and maximized for bin covering. Constraint (2) guarantees that each item is assigned to exactly one bin. Constraint (3) guarantees that the value of each used bin is on or below ($\bowtie$ is $\leq$) the target value t for bin packing, and on or above ($\bowtie$ is $\geq$) the t for bin covering. Constraints (4) and (5) define the domains of the decision variables.

For bin covering, Constraint (2) can actually be relaxed to ≤ 1, since not all values are necessarily placed in some bin. However, such *surplus* values (see below) cannot constitute a bin on their own, and since there is no upper bound on the bins, the surplus values may be put on any bin without altering the optimal solution. In fact, with a rigorous

definition of the surplus values, we can always remove the surplus values from the optimal solution, and the surplus-free solution will still be optimal.

B. The Subset Formulation

A less trivial approach to formulate the bin sorting problem as a MILP problem is to enumerate the prospective bins by sort the items into *packages* that fulfill the target constraint, and then formulate a problem of choosing the smallest or largest number of such packages. If we generate *all possible* such packages, the MILP-solver will have freedom enough to find the optimal number of packages.

Let $p_j \subseteq V$ be a package such that $\sum_{v_i \in p_j} v_i \bowtie t$. Note that contrary to bins, different packages may share items, that is, for some packages p_i and p_j with $i \neq j$ it may hold that $p_i \cap p_j \neq \emptyset$. Let $P_i = \{p_j | v_i \in p_j\}$ be the set of all packages that include the item v_i. We call the elements of P_i *overlapping*, and there are n such sets.

Given the set of k generated packages, and with a slight abuse of notation we use p_j to denote a 0-1-variable representing whether the package p_j is used ($p_j = 1$) or not ($p_j = 0$), we can formulate the MILP problem as:

$$\min / \max \sum_{j=1}^{k} p_j \text{ subject to} \tag{6}$$

$$\sum_{p_j \in P_i} p_j = 1 \qquad \forall i = 1, \ldots, n \tag{7}$$

$$p_j \in \{0, 1\} \qquad \forall j = 1, \ldots, k \tag{8}$$

Similarly to (1), the objective function (6) sums over all the variables representing whether a package is used or not, and since these are 0-1-variables, the sum is the number of used packages; this sum is to be minimized, or maximized, for bin packing and bin covering, respectively. Constraint (7) guarantees that exactly one of the overlapping packages is used, which prohibits multiple inclusion of the same item into the optimal solution, and so guarantees that the chosen set of packages partition V (this is what makes the chosen packages bins). Again for the bin covering the constraint may be relaxed into a less then equality, since the maximisation will make sure that as many packages as possible are chosen; on the other hand, without the more restrictive constraint, the bin packing problem would always yield a solution counting zero bins. Constraint (8) simply defines the domains of the p_j variables.

C. Equivalence class formulation

Though the subset formulation of Section II-B goes a long way to mitigate the computational complexity, observing that for large bin sorting problems we can have, and typically do have, many items with equal values, we can give an even more compact problem formulation, where equal values are not viewed as distinct items, but rather as a single item with a multiplicity equal to the number of actual such same-valued items. This is done by collecting equal items into *equivalence classes*, and instead of enumerating each item of such a class, formulate the optimization problem over integer decision variables related to the number of items in each equivalence class.

Consider a bin sorting problem $\langle V, t, \bowtie \rangle$. Let an equivalence class E_q be a subset of equal valued items of V; that is, $E_q = \{v_i \in V | v_i = w\}$ for a fixed value w. Obviously, $\min(E_q) = w$. Let p denote the number of all equivalence classes. The set of p equivalence classes partition V.

We call a tuple $\langle E_q, f_q \rangle$ of an equivalence class E_q and a *factor* f_q, a *selection*. The factor is used to denote the number of items from the equivalence class that are selected in a certain situation. Of course, $0 \leq f_q \leq |E_q|$, and obviously there is a finite number of distinct selections.

Let a *package class* $PC_i = \{\langle E_1, f_{1,i} \rangle, \ldots, \langle E_p, f_{p,i} \rangle\}$ be a set of selections over all equivalence classes, such that

$$\sum_{\langle E_q, f_{q,i} \rangle \in PC_i} \min(E_q) \cdot f_{q,i} \bowtie t, \tag{9}$$

where $\bowtie$ is $\leq$ for bin packing and $\geq$ for bin covering, and the objective is to minimize and maximize, for packing and covering, respectively. Let k denote the number of all possible package classes.

Since all package classes contain all equivalence classes, albeit many with a zero factor, all package classes overlap, which then becomes an uninteresting observation (contrary to Section II-B).

Given a set of k generated package classes, and with some abuse of notation let PC_i be an integer that represents how many "instances" of the package class PC_i that are included in the optimal solution, then the optimization problem can be formulated as:

$$\min / \max \sum_{i=1}^{k} PC_i \text{ subject to} \tag{10}$$

$$\sum_{i=1}^{k} f_{q,i} \cdot PC_i = |E_q| \qquad \forall q = 1, \ldots, p \tag{11}$$

$$PC_i \geq 0 \qquad \forall i = 1, \ldots, k \tag{12}$$

The objective function (10) sums over all the variables representing the number of each package class instance; this sum is to be minimized for bin packing and maximized for bin covering. Constraint (11) sums for each equivalence class over all the package classes and multiplies with the factor for each respective package class, to get the total number of used values from the specific equivalence class. Naturally, this number cannot exceed the number of values in the equivalence class for the package class, and has to be exactly equal to the number of values in the equivalence class in order to avoid trivial solutions with zero bins. Constraint (12) simply sets zero as the lower bound for the number of instantiations of the respective package classes.

An upper bound for PC_i could be pre-calculated, but it is not clear whether this will have any impact on the computational complexity, and this has not been investigated.

For brevity, we will in the following use the term *package* in place of *package class*.

III. Optimal Solutions

Though theoretically possible, generating *all* packages is practically intractable; we can do this only for small n. And though generating *all* possible package classes is more tractable than generating all packages, it still amounts to a huge computational effort for large bin sorting problems.

However, we can calculate the packages in a more clever way, by observing that the optimization criterion really means that the resulting bins of an optimal solution should be as close to the target value as possible. Calculating such packages saves a lot of computational effort, as is shown in Section V.

In this section we will argue for why and how we can calculate a specific subset of all possible packages, but still get an optimal solution from the subset and equivalence class formulations of Section II. For clarity, we will here treat packing and covering separately in their own subsections.

In both cases, we have a bin sorting problem $\langle V, t, \leq \rangle$ for packing, and $\langle V, t, \geq \rangle$ for covering. The total value over all n items of V is $W = v_1 + v_2 + \cdots + v_n$. A *feasible* solution $f_t = \{b_1, b_2, \ldots, b_m\}$ to a bin sorting problem, is a solution where for each bin, $B_j \leq t$ for packing, and $B_j \geq t$ for covering, and an *optimal* solution is a feasible solution where the number of bins m is minimized for packing, and maximized for covering.

Let us also note the distinction between *bins* and *packages*. Bins partition V so that no item in V can be in more than one bin, whereas packages can share items; it is the task of the bin sorting solver to select among the packages so that a single item does not appear in more than one bin.

A. Bin Packing

A feasible solution $f_t = \{b_1, b_2, \ldots, b_m\}$ for a bin packing problem $\langle V, t, \leq \rangle$ is said to be *true* if all items of V are sorted.

For a feasible solution, all bins b_j are on or below target, that is $B_j \leq t$. For bin packing we want to minimize the number of bins. Thus, it seems to make sense to have bins that are as close to the target (but not above) as possible.

Definition 1: A bin (or package) is said to be *fit* if adding the least valued item from *V* gets it above the target value. That is, a bin b_j is fit if

$$B_j + \min(V) > t.$$

Definition 2: Let V^v be a set of *virtual* items, such that $V^v \cap V = \emptyset$ and $v_k = \min(V)$ for all $v_k \in V^v$.
The virtual items are not in the original problem formulation but are introduced as possible items that can be put in bins to complete a bin into a fit bin, in order to guarantee fit solutions. However, we show later that virtual items can be removed from a solution and thus a *true* solution can be generated.

A feasible solution $f_f = \{b_1, b_2, \ldots, b_m\}$ for a bin packing problem $\langle V, t, \leq \rangle$ is said to be *fit* if all items of V, plus an arbitrary number of virtual items, are sorted, and all bins in f_f are fit.

Lemma 1: For a bin packing problem $\langle V, t, \leq \rangle$, a feasible *true* solution f_t exists if and only if a feasible fit solution f_f exists.

Proof: First we show that to a feasible true solution f_t we can add virtual items to non-fit bins to make a feasible fit solution f_f.

Assume a non-fit bin b_j, thus $B_j + \min(V) \leq t$. Add $\lfloor (t - (B_j + \min(V)))/\min(V) \rfloor$ number of virtual items to b_j, the bin is now fit by definition since adding one additional virtual item makes the value of the bin be larger than the target value t. This can be done for any non-fit bin in f_t.

Second, we show that we can remove (virtual) items from the bins of f_f and get feasible *true* solution.

Assume a fit solution f_f. Let B^f be the sum of the values of all bins for a feasible fit solution f_f. Then the total number of virtual items is equal to $\lfloor (B^f - W)/\min(V) \rfloor$. If this number of virtual items are removed from the bins in f_f, the total number of items of value $\min(V)$ in all bins will be equal to $|E_n|$ and thus the modified solution will be *true*. Note that removing virtual items from a bin can only decrease its value, and since a fit bin is by definition already below the target, so will the *reduced* bin be. The total number of items of value $\min(V)$ will be equal to or larger than $\lfloor (B^f - W)/\min(V) \rfloor$, thus it is possible to remove $\lfloor (B^f - W)/\min(V) \rfloor$ items of value $\min(V)$. Consider any set of bins that is the result of removing $\lfloor (B^f - W)/\min(V) \rfloor$ items of value $\min(V)$ from the bins of the fit solution f_f. The set of reduced bins result in a feasible solution for the *true* problem, since all bins have a value less than the target value and the number of items for every value will be equal to the number of values in the equivalence class for the same value. ■

Theorem 1: Let B^t_{opt} be an optimal true solution to the bin packing problem $\langle V, t, \leq \rangle$, and let B^f_{opt} be an optimal fit solution to the same problem. Then

$$|B^t_{opt}| = |B^f_{opt}|.$$

Proof: If there exists a feasible solution for the fit problem that is optimal, then no better solution than that exists and, according to Lemma 1, there must exist a feasible solution for the *true* problem that yields the same result and thus no better solution can exist. ■

B. Fit package generation

As mentioned, a major issue with the equivalence classes approach to solve bin packing problems is the computation of all package classes that might form part of the optimal solution. Computing and then filtering the power set of V is computationally heavy even for relatively small size problems, therefore a less demanding procedure is required. Given the aforementioned notions, the computation of all *fit* package classes can be formulated as a Constraint Satisfaction Problem (CSP).

Regard the bin packing problem $\langle V, t, \leq \rangle$, with the equivalence classes E_q $(q = 1, \ldots, p)$. Let f_q $(q = 1, \ldots, p)$ be the factor for E_q, that is, an integer variable representing how many values from E_q that are chosen to form a package class. Let F be an integer number such that $F = \lfloor t/\min(V) \rfloor$. The CSP formulation is as follows:

$$0 \leq f_q \leq F \qquad \forall q = 1, \ldots, p \quad (13)$$

$$\sum_{\substack{j=1 \\ q \neq j}}^{p} \Big(\min(E_j) \cdot f_j + (f_q + 1) \cdot \min(E_q) \Big) > t \qquad \forall q = 1, \ldots, p \quad (14)$$

Constraint (13) limits the factor for each equivalence class to be at most as large as the value F; constraint (14) states that the sum of values contained in a *fit package* goes above the target value as soon as we increase the value of any factor by *one*.

Finding a satisfiable solution to this CSP problem means to find a combination of values that, together will result in a *fit* package class. To obtain the whole set of fit package classes, it is possible to set up another problem with the same constraints, plus one constraint ruling out the solution just found. Let $S = \{f_1^*, \ldots, f_p^*\}$ be the solution of the CSP problem, where f_i^* $\forall i = 1, \cdots, p$ is the factor selected for the equivalence class p, then the additional constraint is:

$$\neg\Big(\bigwedge_{f_i^* \in S} (f_i = f_i^*)\Big) \tag{15}$$

Constraint 15 ensures that the solution found in the previous iteration cannot be selected in the current one, so that the solver has to find a new one. In order to find the complete set of *fit* package classes, one has to set up a loop and, for each iteration, add the constraint to rule out the last solution found for the new CSP problem. The loop goes on until the problem becomes unsatisfable, which means that all package classes have been found.

Example of Equivalence Class Formulation: Consider the bin packing problem $\langle V, t, \leq\rangle$, with $V = \langle 50, 50, 40, 40, 10, 10\rangle$ and $t = 100$.

There are three equivalence classes, all of size 2:

$E_1 = \langle 50, 50\rangle$
$E_2 = \langle 40, 40\rangle$
$E_3 = \langle 10, 10\rangle$

And there will also be 8 virtual items of value 10. We need to be able to form feasible packages by using the CSP model in Section III-B and, since we only have two items of value 10 and the target is 100 we need 8 virtual items.

The CSP will now yield all packages that are "just below the target", meaning that if the smallest items from V is added, namely item of value 10, the total value will outreach the target value 100

$PC_1 = \{\langle E_3, 10\rangle\}$
$PC_2 = \{\langle E_2, 1\rangle, \langle E_3, 6\rangle\}$
$PC_3 = \{\langle E_2, 2\rangle, \langle E_3, 2\rangle\}$
$PC_4 = \{\langle E_1, 1\rangle, \langle E_3, 5\rangle\}$
$PC_5 = \{\langle E_1, 1\rangle, \langle E_2, 1\rangle, \langle E_3, 1\rangle\}$
$PC_6 = \{\langle E_1, 2\rangle\}$

For readability, only the equivalence classes E_q with factors $f_q > 0$ have been included above. Now, regarding only the non-zero factors for each equivalence class in the package classes, constraint (11) becomes:

$$\begin{aligned} &PC_4 + PC_5 + 2\ PC_6 \geq 2 \\ &2PC_2 + 2\ PC_3 + PC_5 \geq 2 \\ &10\ PC_1 + 6\ PC_2 + 2\ PC_3 + 5\ PC_4 + PC_5 \geq 2 \end{aligned} \tag{16}$$

We only have two items of value 10 and, in PC_4 and PC_5, E_3 has multiplicity 1, while in PC_6 E_3 has multiplicity 2, hence the coefficients in the first inequality. The bin packing requires that all items are placed into bins, therefore there should be an equality sign; when we generated packages though, we used virtual bins also, therefore now we need to allow for more items than actually available in V, but not less, hence the "$\geq$" sign. The same procedure applies to the other two equivalence classes as shown in second and third inequality. Since the problem is a minimization, most of the virtual items will not be used (the ones left are not enough to form another bin), therefore the solution will be optimal with respect to V (as explained in detail later on).

Then, the objective is:

$$\min PC_1 + PC_2 + PC_3 + PC_4 + PC_5 + PC6.$$

C. Bin Covering

For bin covering we want to maximize the number of bins, and hence it seems to make sense to have bins that are as close to the target (but not below) as possible.

Definition 3: A bin (or package) is said to be *skinny* if removing from it its least valued item gets it below the target value. That is, a bin b_j is skinny if

$$B_j - \min(b_j) < t.$$

A feasible solution $f_s = \{b_1, b_2, \ldots, b_m\}$ for a bin covering problem $\langle V, t, \geq\rangle$ is said to be *skinny* if all bins in f_s are skinny, and an arbitrary number of items of V is sorted. The items of V that are not sorted are called the *surplus* items. A feasible solution $f_t = \{b_1, b_2, \ldots, b_m\}$ to a bin covering problem $\langle V, t, \geq\rangle$, is said to be *true* if all items are sorted.

Lemma 2: For a bin covering problem $\langle V, t, \geq\rangle$, a feasible *true* solution f_t exists if and only a feasible skinny solution f_s exists.

Proof: Regard a true feasible solution $f_t = \{b_1, b_2, \ldots, b_m\}$. For each non-skinny bin b_j, we can iteratively remove its least valued item $\min(b_j)$ until b_j becomes skinny. Obviously, the resulting skinny solution will have the same number of bins as f_t.

Regard a skinny feasible solution $f_s = \{b_1, b_2, \ldots, b_m\}$. This has a set of non-sorted *surplus* items, V_{sur}. These items can be put on arbitrary skinny bins to make the bins non-skinny. Doing so for a bin $b_j \in f_s$ will increase the value B_j which means that it is still on or above target. Thus, we can from the skinny feasible solution generate a true solution f_t.

For both implications, the number of bins is preserved. ■

Theorem 2: Let B_{opt}^t be an optimal true solution to the bin covering problem $\langle V, t, \geq\rangle$, and let B_{opt}^s be an optimal skinny solution to the same problem. Then

$$|B_{opt}^t| = |B_{opt}^s|.$$

Proof: If there exists a feasible solution for the skinny problem that is optimal, then no better solution than that exists and, there must exist a feasible solution for the true problem that yields the same result and no better solution can exist. ■

D. Skinny package generation

As for the bin covering, it is possible to setup a Constraint Satisfaction Problem based on the definition of Fit Bin and run it multiple time to produce, at each iteration, a different valid package, until the problem becomes unfeasible; then we know we have found all feasible packages.

Regard the bin covering problem $\langle V, t, \geq\rangle$, with the equivalence classes E_q $(q = 1, \ldots, p)$. Let f_q $(q = 1, \ldots, p)$ be the factor for E_q, that is, an integer variable representing

how many values from E_q that are chosen to form a package class. The CSP formulation is as follows:

$$0 \leq f_q \leq |E_q| \qquad \forall q = 1, \ldots, p \quad (17)$$

$$f_q > 0 \Rightarrow \sum_{\substack{j=1 \\ q \neq j}}^{p} \Big(\min(E_j) \cdot f_j + (f_q - 1) \cdot \min(E_q) \Big) < t \qquad \forall q = 1, \ldots, p \quad (18)$$

Constraint 17 limits the factor for each equivalence class to be at most as large as the size of the equivalence class itself; constraint 18 states that the sum of values contained in a *skinny package* goes below the target value as soon as we reduce the value of any factor by *one*. Unlike the corresponding constraint for the bin packing problem, in this case it is necessary to specify that we can only reduce a value if it is larger than zero.

Let $S = \{f_1^*, \ldots, f_p^*\}$ be the solution of the CSP problem, where $f_i^* \ \forall i = 1, \cdots, p$ is the factor selected for the equivalence class p, then the additional constraint is:

$$\neg\Big(\bigwedge_{f_i^* \in S} (f_i = f_i^*) \Big) \quad (19)$$

In order to find the complete set of *skinny* package classes, one has to set up a loop and, for each iteration, add the constraint to rule out the last found for the new CSP problem. The loop goes on until the problem becomes unsatisfiable, which means that all package classes have been found.

Example of Equivalence Class Formulation: Consider the bin covering problem $\langle V, t, \geq \rangle$, with V and t as in section III-B and so are the equivalence classes.

We can generate four skinny package classes:

$PC_1 = \{\langle E_1, 2\rangle\}$
$PC_2 = \{\langle E_1, 1\rangle, \langle E_2, 2\rangle\}$
$PC_3 = \{\langle E_1, 1\rangle, \langle E_2, 1\rangle, \langle E_3, 1\rangle\}$
$PC_4 = \{\langle E_2, 2\rangle, \langle E_3, 2\rangle\}$

For readability, only the equivalence classes E_q with factors $f_q > 0$ have been included above. Now, looking only at the non-zero factors for each equivalence class in the package classes, constraint (11) becomes:

$$\begin{aligned} &2\ PC_1 + PC_2 + PC_3 \leq 2 \\ &2\ PC_2 + PC_3 + 2\ PC_4 \leq 2 \\ &PC_3 + 2\ PC_4 \leq 2 \end{aligned} \quad (20)$$

Since the the bin covering does not require all items to be allocated to binsl, the equality constraint can be relaxed, hence the inequalities above.

Finally, the objective is:

$$\max PC_1 + PC_2 + PC_3 + PC_4.$$

One optimal solution is to select PC_1 once ($PC_1 = 1$) and PC_4 once ($PC_4 = 1$). However, another optimal solution is to select two "instances" of PC_3 ($PC_3 = 2$). Both of these are of course skinny solutions.

So, we really only need to generate fit (for packing) and skinny (for covering) packages and package classes to get optimal solutions from the subset and equivalence class formulations. This saves a lot of computational effort, as shown in Section V.

IV. Sub-Optimal Solutions

The equivalence class formulation significantly improves the runtime performance of the optimizer compared to the naïve formulation, as shown in Section V. Nevertheless, BC is still a combinatorial NP-hard problem and for some problems, calculating the true optimum might be expensive in terms of computational effort. The alternative is to consider heuristic methods that can provide a sub-optimal solution in a shorter time. Based on the equivalence classes approach, a heuristic method was developed to simplify the problem and calculate a suboptimal solution faster. One hypothesis that led to the heuristic is that the number of package classes related to a BC problem $\langle V, t, \geq \rangle$ does not depend entirely on $|V|$, but rather on the number of equivalence classes and their cardinality; the more different values, the more possible combinations, the more package classes.

Therefore, the goal of such method is to provide a set of approximated equivalence classes, C^*, where a certain number of consecutive exact equivalence classes are merged together. We call the approximated equivalence classes *chains*.

Let $E = \langle E_1, E_2, \ldots, E_p \rangle$ be the set of p number of equivalence classes over the values of V, ordered so that $\min(E_i) < \min(E_{i+1})$ (for $i = 1, \ldots, p-1$). Let C be a set of *chains*, sets of one or more consecutive equivalence classes from E, $C = \bigcup_{i=1}^{p-l+1} \{E_i, E_{i+1}, \ldots E_{i+l-1}\}$ for $l = 1, \ldots, p$, and let the size of a chain be the sum of the sizes of the equivalence classes it is composed of; for $C_j \in C$, $\|C_j\| = \sum_{E_i \in C_j} |E_i|$.

Note that chains will have common elements (equivalence classes), just as packages, therefore it is possible to define the set O_i of all chains containing a certain value.

Example of Chains Generation: $E = \langle E_1, E_2, E_3, E_4 \rangle$, $p = 4$, with $\min(E_i) < \min(E_{i+1})$ for $i = 1, \ldots, 3$,

$C = \bigcup_{i=1}^{p-l+1} \{E_i, E_{i+1}, \ldots E_{i+l-1}\}$ for $l = 1, \ldots, p$

$\{E_1\}, \{E_2\}, \{E_3\}, \{E_4\}, \qquad l = 1, i = 1, \ldots, p$
$\{E_1, E_2\}, \{E_2, E_3\}, \{E_3, E_4\}, \qquad l = 2, i = 1, \ldots, p-1$
$\{E_1, E_2, E_3\}, \{E_2, E_3, E_4\}, \qquad l = 3, i = 1, \ldots, p-2$
$\{E_1, E_2, E_3, E_4\} \qquad l = 4, i = 1, \ldots, p-3$

A. Heuristic for the bin packing problem

When it comes to the bin packing problem, a way to generate the above mentioned chains is to merge exact equivalence classes as explained in the above section and to assign to all the values of the resulting approximated equivalence class the largest value of all classes that have been merged. This is done so that the solution generated when solving the simplified problem is valid: if a value smaller than the one belonging to the largest class merged were to be assigned to the approximated class, the optimal solution to the simplified problem might be smaller than the optimal solution to the original problem and, therefore, unfeasible.

Definition 4: Let define the *minimum theoretical number of bins M* as the number of bins we could achieve if we could break down the items into smaller ones and reallocate the overweight from each bin to form other ones: $M = {}^{W}/_{t}$. Such value can be achieved by relaxing the integrality

constraint in the initial problem, as pointed out in [11] where such value is defined as *lower bound*.

Let the chain E_{l-m} be the result of merging the equivalence classes E_l, E_m, where $\min(E_l) < \min(E_m)$. Then $\min(E_{l-m}) = \min(E_m), |E_{l-m}| = |E_l| + |E_m|$.

$$|E_{l-m}| \cdot \min(E_{l-m}) = (|E_l| + |E_m|) \cdot \min(E_m) > |E_l| \cdot \min(E_l) + |E_m| \cdot \min(E_m)$$

The gain γ is then:

$$\min(E_m) \cdot (|E_l| + |E_m|) - |E_l| \cdot \min(E_l) + |E_m| \cdot \min(E_m) = |E_l| \cdot (\min(E_m) - \min(E_l))$$

The new minimum theoretical number of bins is $M' = {}^{(W+\gamma)}/_{t}$. The gain can be used to decide which classes is better to merge when simplifying an instance. Using a greedy algorithm it is possible to quickly generate all chains and calculate γ for each of them. It is then possible to formulate a MILP model to select the chains that produce the minimum loss, given a desired number of equivalence classes *d*.

Let x_i $(i = 1, \ldots, k)$ be 0-1-variables representing whether the chain C_i is chosen $(x_i = 1)$ or not $(x_i = 0)$. The MILP formulation is as follows:

$$\min \sum_{i=1}^{k} x_i \cdot \gamma_i \tag{21}$$

$$\sum_{i=1}^{p} x_i = d \tag{22}$$

$$\sum_{x_i \in O_j} x_i = 1 \qquad \forall j = 1, \ldots, p \tag{23}$$

$$x_i \in \{0, 1\} \qquad \forall i = 1, \ldots, k \tag{24}$$

The objective function (21) is the sum of all gains for the chains that are chosen. Constraint 22 states that exactly d chains have to be chosen. Constraint (23) states that overlapping chains are mutually exclusive. Finally, (24) sets the variable domains to be binary.

Note that the heuristic has been developed bearing in mind that, with a normal distribution, there are only a few values at the edges of the *bell curve*, while most of the values appear in the middle of it. Therefore, while containing the same number of values, the chains that are closer to the edges will contain more classes from E, involving a larger loss than the ones closer to the centre. However, as those values are smaller in number compared to the ones in the middle, the overall loss seems not to be significant.

B. Heuristic for the bin covering problem

Once again it is possible to draw inspiration from the results achieved for the bin packing problem to develop a working heuristic method for the bin covering problem too. By merging equivalence classes it is in fact possible to generate a simplified problem that provides a sub optimal solution. This time it is required to assign to the resulting equivalence class the value of the items from the class with the smallest values, in order to generate a valid solution. Since this is a maximization problem, it makes more sense to talk about *loss* σ (rather than a gain) that affects the maximum (instead of minimum) theoretical number of bins. Once again this value corresponds to the optimum of the cost function for the bin covering when relaxing the integrality constraint.

It is possible to calculate the *loss* σ related to the merging of two or more equivalence classes in terms of decrease in the total value *W* and, therefore, of the maximum theoretical number of bins *M*.

Let the chain E_{l-m} be the result of merging the equivalence classes E_l, E_m, where $\min(E_l) < \min(E_m))$. Then $\min(E_{l-m}) = \min(E_l), |E_{l-m}| = |E_l| + |E_m|$.

$$|E_l| \cdot \min(E_l) + |E_m| \cdot \min(E_m) > |E_{l-m}| \cdot \min(E_{l-m}) = \min(E_l) \cdot (|E_l| + |E_m|)$$

The loss σ is then:

$$|E_l| \cdot \min(E_l) + |E_m| \cdot \min(E_m) - \min(E_l) \cdot (|E_l| + |E_m|) = (\min(E_m) - \min(E_l) \cdot |E_m|)$$

The new maximum theoretical number of bins is $M' = {}^{(W-\sigma)}/_{t}$. Setting up exactly the same MILP model as in IV-A it is possible to find the set of merged classes that minimizes the gain while guaranteeing a desired number of equivalence classes for the simplified problem.

Example of Chain Optimization: Consider the example shown in Section IV, and assume that the values and sizes of the equivalence classes are respectively $\min(E_1) = 13$, $|E_1| = 10$, $\min(E_2) = 15$, $|E_2| = 7$, $\min(E_3) = 20$, $|E_3| = 12$, $\min(E_4)$, $|E_4| = 3$. If the desired number of classes is $d = 2$ we can compute the loss for each chain based on (21)–(24):

$C_1 = \{E_1\}$ $\qquad \sigma(C_1) = 0$

$C_2 = \{E_2\}$, $\qquad \sigma(C_2) = 0$

$C_3 = \{E_3\}$, $\qquad \sigma(C_3) = 0$

$C_4 = \{E_4\}$, $\qquad \sigma(C_4) = 0$

$C_5 = \{E_1, E_2\}$, $\qquad \sigma(C_5) = 7 \cdot (15 - 13) = 14$

$C_6 = \{E_2, E_3\}$, $\qquad \sigma(C_6) = 12 \cdot (20 - 15) = 60$

$C_7 = \{E_3, E_4\}$, $\qquad \sigma(C_7) = 3 \cdot (25 - 20) = 15$

$C_8 = \{E_1, E_2, E_3\}$,
$\sigma(C_8) = \sigma(C_5) + 12 \cdot (20 - 13) = 98$

$C_9 = \{E_2, E_3, E_4\}$,
$\sigma(C_9) = \sigma(C_6) + 3 \cdot (25 - 15) = 90$

$C_{10} = \{E_1, E_2, E_3, E_4\}$,
$\sigma(C_{10}) = \sigma(C_8) + 3 \cdot (25 - 13) = 134$

According to constraint (22) only *d* chains can be selected:

$$C_1 + C_2 + C_3 + C_4 + C_5 + C_6 + C_7 + C_8 + C_9 + C_{10} = 2$$

According to constraint (23) some chains are mutually exclusive, so each equivalence class must be picked exactly once. As stated before, chains are sets, but with some abuse of notation, we here use them as binary variables to state whether a chain is selected $(C_i = 1)$ or not $(C_i = 0)$.

$$C_1 + C_5 + C_{10} = 1$$
$$C_2 + C_5 + C_6 + C_8 + C_9 + C_{10} = 1$$
$$C_3 + C_6 + C_7 + C_8 + C_9 + C_{10} = 1$$
$$C_4 + C_7 + C_9 + C_{10} = 1$$

Finally, the objective function to minimize is:

$$C_1 \cdot \sigma(C_1) + C_2 \cdot \sigma(C_2) + C_3 \cdot \sigma(C_3) \\ +C_4 \cdot \sigma(C_4) + C_5 \cdot \sigma(C_5) + C_6 \cdot \sigma(C_6) + C_7 \cdot \sigma(C_7) \\ +C_8 \cdot \sigma(C_8) + C_9 \cdot \sigma(C_9) + C_{10} \cdot \sigma(C_{10})$$

V. Computational Analysis

We ran an extensive analysis over 1500 generated problems, comparing both the standard and the equivalence class formulations for both the bin packing and bin covering problem; we also compare the standard and equivalence classes formulations for the bin packing problem over different benchmark sets from the literature:

- Falkenauer [14]: two sets with 80 instances each;
- Scholl [15] three sets with 720, 480 and 10 instances respectively;
- Schwerin [16] two sets with 100 instances each;
- Wäscher [17] a set with 17 instances;
- Schoenfield [18] a set with 28 instances.

All instances have been solved using the state-of-the-art MILP solver Gurobi 9 [19]. The time limit is 1200 seconds and the solver has been used with its default setting. All the experiments were performed on an *Intel Core i7 6700K, 4.0 GHZ, 32GB RAM* running *Ubuntu-16.04*. The implementation of all the models presented in this paper, as well as the benchmark instances are available on https://github.com/sabinoroselli/bin_covering-packing.git

To generate the instances we approximated a normal distribution. The parameters to generate instances are the number of items, the range of values, the average value of such range (which is the mean of the distribution) and the standard deviation. Since the results show a skewed distribution, we reported the median and upper and lower quartile for each category.

The first set of tests, reported in Table III has been run using the standard formulation to solve both the covering and the packing problem. The number of items ranges from 10 to 70 and the target value ranges from 300 to 900. The values of the items range from 130 to 170 and the value of the deviation ranges from 10 to 90; finally, five different instances are generated for each set of parameters by changing the random generation seed. Results show that different values for the deviation do not affect the running time significantly so they are not shown explicitly in the table: for each size and target value, the average over 25 instances is shown (five different values of deviation times five different random seeds).

Results from this first simulation show that, as expected, an increasing number of items makes the problem harder to solve. It is interesting though, to notice that even for relatively high number of items, there are still many instances that can be solved almost instantly which means that also large problems can have trivial solutions. What does affect the complexity of the instance, according to the results, is the target value t; though there are some outliers, most of the unsolved instances for the covering problem have a target value of 500, while for the packing problem it is 700.

In the second set of tests the number of items ranges from 60 to 500, while the other parameters are the same as in the previous tests. For the instance classes counting 200 to 500 items and with a target value of 800 and 900 it was only possible to solve one instance, therefore it was not possible to calculate the quartiles. The results, summarized in Table IV show, as for the tests run for the standard formulation, an increasing amount of time required to find the optimal as the number of items and the target value increases (given a certain distribution and, thus, a certain number of equivalence classes). Unlike the standard approach though, the equivalence classes formulation seems to have a more steady trend: the former's performance is heavily affected by the intrinsic hardness of a specific instance, being able sometimes to immediately solve large size instances with a large target value, while getting stuck on relatively small sized instances; on the other hand, the latter's behaviour is more predictable and steadily increases with the instances parameters, as shown in Figure 1. Also, the comparison of Table III and Table IV show tighter values of the quartiles for the equivalence classes formulation; this mean that the solving time for a given class of instances (in terms of generated packages for example) is more predictable.

Another conclusion we can draw from the data is the strong correlation between the solving time and the number of packages, which in turn is affected by the target value and, to a lesser extent, by the number of objects. A larger target value means an exponentially higher number of possible combinations to form skinny (or fit) packages; also having a higher number of items in each equivalence class means that it is possible to form more combinations of them that make valid packages, though this is true only up to a certain value. For instance, if the target value is 300, having two or 200 items of value 200 does not make any difference.

For this reason, an instance with 60 items and a target value of 300 only yields a few thousand packages, while the same instance with a target value of 900 can count millions of them. As mentioned before, the increase in the number of items also causes an increase in the number of packages, which seems nonetheless to happen within the same order of magnitude.

Though the number of packages has a direct impact on the solving time, the equivalence class formulation still proves to be efficient to solve very large sized problems, being scalable in terms of number of items (which is usually the limitation

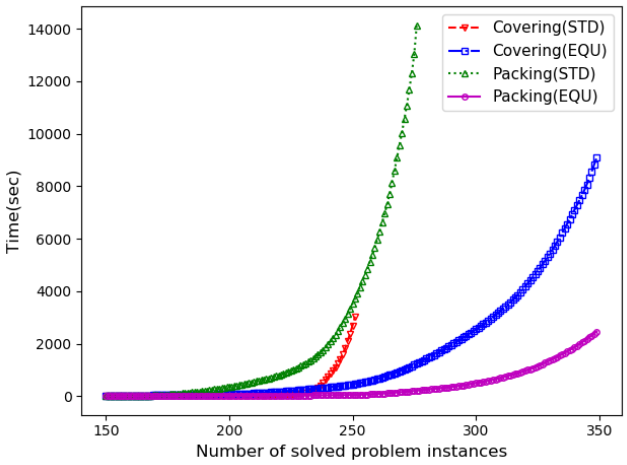

Fig. 1: Comparison of model formulations (STD stands for *standard* and EQU for *equivalence classes*) and sorting type over generated problems.

TABLE I: Comparison of the standard formulation and the equivalence class formulation over the benchmark instance sets, showing the number of solved instances within one minute and the average time calculated over the instances that did not time out. For each instance set it is reported the total number of instances and the number of instances that generate less than ten million packages. The '-' is used when no instance is solved, while the '*' means that the package generation algorithm run out of memory.

	Instances		STD		EQU	
	Complete Set	$\leq 10 \times 10^6$	solved	average	solved	average
Falkenauer U	80	80	16	34.11	80	1.64
Falkenauer T	80	80	10	30.07	80	0.76
Scholl 1	720	313	185	26.92	294	1.29
Scholl 2	480	54	50	6.86	48	7.69
Scholl 3	10	7	0	-	0	-
Schwerin 1	100	100	51	32.23	26	48.49
Schwerin 2	100	100	40	37.72	5	53.66
Wäscher	17	0	0	-	*	*
Schoenfield	28	0	0	-	*	*

for the standard approach). Even so, the package generation time via a greedy algorithm and the model generation time are directly proportional to the number of generated packages and for very hard instances they might not be negligible. Figure 2 shows how the package generation time increases with respect to the number of packages (benchmark instances from the *Scholl* set have been employed to evaluate the package generation efficiency): it is still close to one minute for instances leading to four million packages but it goes up to about fifteen minutes for instances of around ten million packages.

The driving force behind the package generation computation time is its space complexity, i.e the amount of memory space required by the algorithm. Allocating, writing and reading to memory takes time. The standard form of a MILP model assumes constraints of the form $\mathbf{Ax} \leq \mathbf{b}$, represented by (11) in our model. Thus, k packages generated from q equivalence classes requires the allocation of $q \times k$ integers to store the final constraint. This implies that the space complexity of the package generation, without considering the implementation, is at least $\mathcal{O}(kq)$ if generating a model in standard form. Since the number of possible packages k grows exponentially with larger target values and the number of unique items, quickly outgrowing the number

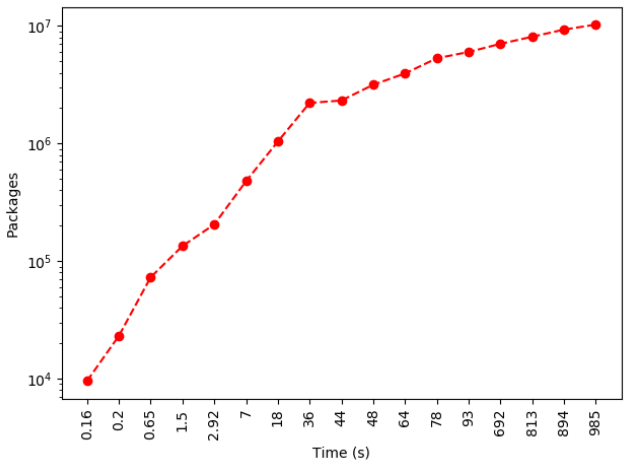

Fig. 2: Evaluation of package generation time (in seconds) against number of generated packages (log scale).

of equivalence classes, the space complexity constitutes a major challenge with larger instances.

For example, the largest benchmark instances we could solve before running out of memory counted around thirty million packages. Many instances included more than one hundred equivalence classes. Storing the constraint coefficients as 32 bit integers would in those cases require $3e^7 \cdot 100 \cdot 4 = 12e^9$ byte or 12 gigabyte.

Potential improvements to the package generation can be divided into two types: (i) reduction of the number of generated packages and (ii) reduction of allocated memory for each package. The first type is by far the strongest and will improve all parts of the process, not only the package generation itself. This can even be considered the main target of the whole equivalence classes approach and especially with the formulation of skinny/fit packages in Section III and the heuristics in Section IV. The second type of improvement depends on the specific implementation of the model and the package generation. The idea is to reduce the memory allocation by formulating a more condense optimization model. Constraints can often be transformed such that they are represented with less coefficients.

Though we have not investigated the subject, we believe that there is room for improvement of the implementation in future work. For example, consider (16) and (20) where we, for readability, only included non-zero elements. This is one way to represent the same constraint with less coefficients. These types of more condense formulations can be done also in the actual optimization model. Another interesting idea is to reduce the memory allocation of the packages using more efficient data structures. The current package generation constructs a list where each package is represented by its equivalence classes, individually. One could instead generate a tree structure that represents all packages, where each node is an equivalence class and branches indicate how many of this class to use in the final package. The depth of such tree would be equal to the number of equivalence classes and each leaf would constitute an unique package. Preliminary experiments has shown that this method greatly decreases the memory allocation of the package generation for some of the benchmark instances, allowing much larger instances to be processed and decreasing the computation time. Future work is required to incorporate such tree structure in the subsequent optimization and to further evaluate the improvement to the space complexity on a larger set of instances.

Table I shows the performance of the equivalence classes method over different sets of benchmark instances. In order to compare our formulation to the other relevant algorithms we found in the literature, we refer to Table I and Table II of [3], where such algorithms are evaluated using the same benchmark sets listed above, and setting the time limit to one minute (the implementation for such algorithms is available at the authors web page [20]). Though the computers used are different, we believe that the comparison still gives a hint of the method's potential. Nonetheless, we decided to run the benchmark instances again for the standard formulation (called *Basic ILP* in [3], since the authors used a different solver). Moreover, the algorithms listed in those tables, are specifically tailored to solve the bin packing problem, while our formulation provides a linear program that can be fed to any solver (or extended with additional constraints) and so can be done with the

standard formulation. Therefore we decided to make a more rigorous comparison between the standard formulation and the equivalence classes formulation.

As already discussed when commenting on Table IV, the number of generated packages directly affects the solving time when using the equivalence classes approach; therefore we only considered instances counting less than ten million packages. Table I shows the number of instances in each set and, next to it, the number of instances that lead to generate less than ten million packages. The equivalence class formulation is much faster than the standard formulation for the *Falkenauer* sets and can deal with all instances in less than 2 seconds each. When it comes to the *Scholl* instances, in the first set the equivalence class formulation is still much faster and can deal with almost twice as many instances before the time limit, while it performs very similarly, though slightly worse, in the second set. Neither method can deal with any of the instances in the third set. The standard formulation performs considerably better than the equivalence class formulation in both *Schwerin* sets, both in terms of instances solved and average time. Finally, the standard formulation cannot solve any of the instances from either the *Wäscher* or the *Schoenfield* set within the time limit, while the equivalence class formulation cannot even get started, since the computer ran out of memory while generating the packages.

A possible remedy to handle such hard instances is to reduce the number of classes by merging them into chains as described in sections IV-B and IV-A. The heuristic does not guarantee an optimal value to the original problem, but it can drastically reduce the number of packages, thus speeding up the model generation tremendously, while producing close-to-optimal results. Table II shows an example for both the covering and the packing problems where the number of equivalence classes is progressively reduced from the original number, shown on the first line, which would give the true optimal solution. The instance is evaluated considering two different target values: 450 is an exact multiple of 150 (the mean value of the items) while 525 is as far as possible from being an exact multiple. Also, two different values for the deviation are selected: 10 (corresponding to the data shown in the upper part of Table II) and 90 (corresponding to the remaining data); neither of the two parameters seem to largely affect the accuracy. What we can see though, is a dramatic reduction in the number of generated packages as the number of classes decreases, while the optimal value for the simplified instances is still close to the optimum for the original one.

VI. Conclusions

In this paper we have presented alternative formulations to solve both the bin covering and the bin packing problem; we have shown that these formulations perform particularly well when the number of different values in the problem instance is limited. In such cases, our formulations allow to solve problems counting hundreds of items in a considerably short time. This feature can prove useful for industrial applications where the number of items is high but the range of values is limited, such as battery recycling (for bin packing) or fixed tray weight sorting in food processing (bin covering). When this is not the case, our formulations allow for problem simplification by means of merging equivalence classes that still provides a close to optimal solution, while dramatically reducing the computation time. Moreover, the concept of skinny/fit packages can constitute a solid base to improve existing specific-purpose algorithms such as those given by [3], which we intend investigate further.

Acknowledgements

The authors thank Folkmar Frederik Ramcke for implementing the algorithms for package generation.

References

[1] E. G. Coffman, J. Csirik, G. Galambos, S. Martello, and D. Vigo, "Bin packing approximation algorithms: survey and classification," in *Handbook of combinatorial optimization*. Springer New York, 2013, pp. 455–531.

[2] S. F. Assmann, D. S. Johnson, D. J. Kleitman, and J.-T. Leung, "On a dual version of the one-dimensional bin packing problem," *Journal of algorithms*, vol. 5, no. 4, pp. 502–525, 1984.

[3] M. Delorme, M. Iori, and S. Martello, "Bin packing and cutting stock problems: Mathematical models and exact algorithms," *European Journal of Operational Research*, vol. 255, no. 1, pp. 1–20, 2016.

[4] G. Belov and G. Scheithauer, "A branch-and-cut-and-price algorithm for one-dimensional stock cutting and two-dimensional two-stage cutting," *European Journal of Operational Research*, vol. 171, no. 1, pp. 85 – 106, 2006.

[5] F. Brandão and J. P. Pedroso, "Bin packing and related problems: General arc-flow formulation with graph compression," *Computers & Operations Research*, vol. 69, pp. 56 – 67, 2016.

[6] F. Clautiaux, S. Hanafi, R. Macedo, M. Émilie Voge, and C. Alves, "Iterative aggregation and disaggregation algorithm for pseudo-polynomial network flow models with side constraints," *European Journal of Operational Research*, vol. 258, no. 2, pp. 467 – 477, 2017.

[7] M. Delorme and M. Iori, "Enhanced pseudo-polynomial formulations for bin packing and cutting stock problems," *INFORMS Journal on Computing*, vol. 32, no. 1, pp. 101–119, 2020.

[8] M. Dell'Amico, F. Furini, and M. Iori, "A branch-and-price algorithm for the temporal bin packing problem," *Computers & Operations Research*, vol. 114, pp. 1 – 16, 2020.

[9] S. Roselli, F. Hagebring, S. Riazi, M. Fabian, and K. Åkesson, "On the use of equivalence classes for optimal and sub-optimal bin covering," in *2019 IEEE 15th International Conference on Automation Science and Engineering (CASE)*. IEEE, 2019, pp. 1004–1009.

[10] L. V. Kantorovich, "Mathematical methods of organizing and planning production," *Management science*, vol. 6, no. 4, pp. 366–422, 1960.

[11] M. Labbé, G. Laporte, and S. Martello, "An exact algorithm for the dual bin packing problem," *Operations Research Letters*, vol. 17, no. 1, pp. 9–18, 1995.

[12] M. Peeters and Z. Degraeve, "Branch-and-price algorithms for the dual bin packing and maximum cardinality bin packing problem," *European journal of operational research*, vol. 170, no. 2, pp. 416–439, 2006.

[13] M. J. Brusco, H. F. Köhn, and D. Steinley, "Exact and approximate methods for a one-dimensional minimax bin-packing problem," *Annals of Operations Research*, vol. 206, no. 1, pp. 611–626, Jul 2013.

[14] E. Falkenauer, "A hybrid grouping genetic algorithm for bin packing," *Journal of heuristics*, vol. 2, no. 1, pp. 5–30, 1996.

[15] A. Scholl, R. Klein, and C. Jürgens, "Bison: A fast hybrid procedure for exactly solving the one-dimensional bin packing problem," *Computers & Operations Research*, vol. 24, no. 7, pp. 627–645, 1997.

[16] P. Schwerin and G. Wäscher, "The bin-packing problem: A problem generator and some numerical experiments with FFD packing and MTP," *International Transactions in Operational Research*, vol. 4, no. 5-6, pp. 377–389, 1997.

TABLE II: Comparison of the optimal solution and solving time (in seconds) with respect to the number of generated packages for an instance of 200 items solved with both packing and covering method when varying the number of classes (Cl.) by simplifying the original instance. The instance is evaluated for target values of 450 and 525 and deviation values of 10 (top part of the table) and 90 (bottom part).

Cl.	COVERING						PACKING					
	Standard Deviation: 10											
	450			525			450			525		
	Packages	Optimum	Time	Packages	Optimum	Time	Packages	Optimum	Time	Packages	Optimum	Time
45	62999	65	0.47	181800	50	2.10	8964	67	0.08	15969	65	0.26
40	44616	65	0.35	118894	50	1.43	6768	67	0.07	11525	65	0.20
35	26057	65	0.19	72368	50	0.50	4437	67	0.04	7814	65	0.12
30	15931	65	0.10	39935	50	0.28	2486	67	0.03	4955	66	0.03
25	8154	65	0.08	19788	50	0.13	1566	67	0.02	2925	66	0.02
20	3742	65	0.03	8621	50	0.05	807	67	0.00	1540	66	0.01
15	1512	65	0.01	3061	50	0.02	411	67	0.00	681	66	0.00
10	445	64	0.00	715	50	0.00	106	68	0.00	220	66	0.00
	Standard Deviation: 90											
	450			525			450			525		
	Packages	Optimum	Time	Packages	Optimum	Time	Packages	Optimum	Time	Packages	Optimum	Time
80	540384	66	9.09	2151942	56	53.90	47068	67	0.52	219327	57	3.77
75	438155	66	6.40	1683682	56	41.53	38926	67	0.37	173769	57	5.44
70	315242	65	3.77	1222000	56	33.51	33502	67	0.25	143324	57	2.24
65	214423	65	1.89	825952	56	15.09	25168	67	0.18	99023	57	6.27
60	181567	65	1.90	647241	56	13.17	18370	67	0.09	67044	57	2.02
55	130228	65	1.26	450641	56	8.11	14857	67	0.08	52659	57	1.19
50	82868	65	0.69	293594	56	2.56	10848	67	0.07	36831	57	0.73
45	67720	65	0.44	216105	56	3.28	7912	67	0.07	25138	57	0.47
40	40267	65	0.23	128418	56	1.95	5988	67	0.04	18789	57	0.16
35	25224	65	0.19	77416	56	0.76	3583	67	0.02	10421	57	0.21
30	14716	65	0.09	42137	56	0.46	2579	67	0.02	7217	57	0.13
25	7903	65	0.05	21239	56	0.26	1491	67	0.02	3650	57	0.03
20	3635	65	0.03	8834	56	0.05	826	67	0.01	1904	58	0.01
15	1437	65	0.01	3149	55	0.02	310	68	0.00	674	58	0.01
10	344	64	0.00	677	54	0.00	112	69	0.00	213	59	0.00

[17] G. Wäscher and T. Gau, "Heuristics for the integer one-dimensional cutting stock problem: A computational study," *Operations-Research-Spektrum*, vol. 18, no. 3, pp. 131–144, 1996.

[18] J. E. Schoenfield, "Fast, exact solution of open bin packing problems without linear programming. draft," *US Army Space & Missile Defence Command, Huntsville*, vol. 20, p. 72, 2002.

[19] Gurobi Optimization, LLC, *Gurobi Optimizer Reference Manual, version 9*, 2020. [Online]. Available: http://www.gurobi.com

[20] M. Delorme, M. Iori, and S. Martello, "BPPLIB: a library for bin packing and cutting stock problems," *Optimization Letters*, vol. 12, no. 2, pp. 235–250, 2018.

Sarmad Riazi received his M.Sc. in Systems, Control and Mechatronics in 2013 and his Ph.D. degree in 2020, both from Electrical Engineering department of Chalmers University of Technology, Gothenburg, Sweden. He was a guest lecturer at University West, Trollhättan, Sweden in 2019. He is working as a research engineer at AGVE, Sweden. His research interests include integrated, combinatorial and nonlinear optimization methods and their industrial applications, and energy minimization of robotic manipulators and AGVs.

Sabino Roselli was born in Bari, Italy, in 1992. He received a M.Sc. degree in Industrial Engineering from Politecnico di Bari, Bari, Italy, in 2017. Since then, he has been pursuing a Ph.D. degree at the Electrical Engineering Department, Chalmers. His filed of research is optimal scheduling of operations within the industrial context.

Martin Fabian is Professor in Automation and Head of the Automation Research group at the Department of Electrical Engineering, Chalmers University of Technology. His research interests include formal methods for automation systems in a broad sense, merging the fields of Control Engineering and Computer Science. He has authored more than 200 publications, and is co-developer of the formal methods tool Supremica, which implements several state-of-the-art algorithms for supervisory control synthesis.

Fredrik Hagebrig was born in Borås, Sweden, in 1985. He received a M.Sc. degree in Systems, Control and Mechatronics from Chalmers University of Technology, Gothenburg, Sweden, in 2016. Since then, he has been pursuing a Ph.D. degree at the Electrical Engineering Department, Chalmers.

Knut Åkesson is Professor in the Department of Electrical Engineering at Chalmers University of Technology, Gothenburg, Sweden. His main research is in using rigorous methods for analysis of cyber-physical systems. Åkesson holds a M.Sc. in Computer Science and Technology from Lund Institute of Technology, Sweden, and PhD in Control Engineering from Chalmers University of Technology, Gothenburg, Sweden.

TABLE III: Set of generated instances solved using the standard formulation for both the packing and the covering problem. The size of the instances ranges from 10 to 70 items and the target value from 300 to 900. The median time is reported (calculated over the solved instances) as well as the lower and upper quartile and the number of instances that timed out. The timeout is set to 1200 seconds. If no instance for a given category is solved, the cell is marked with the symbol '-'.

COVERING STANDARD

	300				400				500				600				700				800				900			
	med.	low	upp	TO	med.	low	upp	TO	med.	low	upp	TO	med.	low	upp	TO	med.	low	upp	TO	med.	low	upp	TO	med.	low	upp	TO
10	0.00	0.00	0.00	0	0.00	0.00	0.00	0	0.00	0.00	0.00	0	0.00	0.00	0.00	0	0.00	0.00	0.00	0	0.00	0.00	0.00	0	0.00	0.00	0.00	0
20	0.00	0.00	0.02	0	0.00	0.00	0.00	0	0.00	0.00	0.02	0	0.00	0.00	0.00	0	0.00	0.00	0.00	0	0.00	0.00	0.00	0	0.00	0.00	0.00	0
30	0.01	0.01	1.16	4	0.01	0.01	0.01	0	8.59	0.09	9.58	4	0.01	0.01	0.01	0	0.01	0.01	0.01	0	0.01	0.01	0.01	0	0.01	0.01	0.02	0
40	0.06	0.02	0.16	7	0.01	0.01	0.01	0	18.57	13.95	22.74	12	0.01	0.01	0.02	0	0.01	0.01	0.01	0	0.04	0.01	0.08	0	0.01	0.01	0.01	0
50	0.07	0.03	0.15	3	0.46	0.35	0.54	0	35.50	26.22	54.81	11	0.02	0.02	0.02	0	0.02	0.01	0.02	0	0.42	0.12	24.89	6	0.02	0.01	0.02	0
60	0.17	0.12	0.27	15	0.03	0.03	0.03	0	90.25	32.75	116.26	13	0.03	0.02	0.03	0	0.03	0.03	0.10	0	-	-	-	25	0.02	0.02	0.03	0
70	0.17	0.15	0.25	6	0.90	0.04	2.74	0	65.83	12.89	236.56	11	0.04	0.03	0.04	0	0.03	0.03	0.07	4	-	-	-	25	0.03	0.03	0.03	0

PACKING STANDARD

	300				400				500				600				700				800				900			
	med.	low	upp	TO	med.	low	upp	TO	med.	low	upp	TO	med.	low	upp	TO	med.	low	upp	TO	med.	low	upp	TO	med.	low	upp	TO
10	0.00	0.00	0.00	0	0.00	0.00	0.00	0	0.00	0.00	0.00	0	0.00	0.00	0.00	0	0.00	0.00	0.00	0	0.00	0.00	0.00	0	0.00	0.00	0.00	0
20	0.06	0.01	0.10	0	0.05	0.03	0.08	0	0.02	0.00	0.03	0	0.00	0.00	0.00	0	0.00	0.00	0.00	0	0.00	0.00	0.00	0	0.00	0.00	0.00	0
30	0.01	0.01	0.23	0	0.18	0.12	0.26	0	0.08	0.00	0.14	0	0.00	0.00	0.01	0	0.01	0.01	0.04	0	0.00	0.00	0.00	0	0.00	0.00	0.02	0
40	3.85	0.40	30.68	0	0.76	0.49	1.98	0	1.13	0.33	2.09	0	0.01	0.01	0.06	0	1.95	0.12	5.14	15	0.01	0.01	0.01	0	0.01	0.01	0.01	0
50	18.33	8.97	82.04	6	6.76	1.15	21.46	0	3.75	1.00	4.95	0	0.02	0.02	0.02	4	4.56	3.06	9.69	16	0.01	0.01	0.01	0	0.01	0.01	0.01	0
60	39.58	0.10	170.01	11	21.77	3.08	40.42	7	7.61	5.44	13.96	0	0.07	0.04	0.22	0	79.61	18.25	124.53	15	0.01	0.01	0.01	0	0.05	0.03	0.16	0
70	1.02	0.10	44.92	13	473.65	53.34	560.20	8	19.92	16.06	27.06	0	0.04	0.04	0.04	3	153.92	20.90	343.86	13	0.02	0.02	0.02	0	0.02	0.02	0.03	4

TABLE IV: Set of generated instances solved using the equivalence classes formulation for both the packing and the covering problem. The size of the instances ranges form 60 to 500 items and the target value ranges from 300 to 900. The median time is reported as well as the lower and upper quartile and average number of packages generated (up to tens of thousands, numbers are shown in regular notation, afterwards the scientific notation is employed. The timeout is set to 1200 seconds. The symbol '-' means that only one instance for that class was solved, therefore it was not possible to calculate the upper or lower quartile.

COVERING EQUIVALENCE CLASSES

	300				400				500				600				700				800				900			
	med.	low	upp	# pack	med.	low	upp	# pack	med.	low	upp	# pack	med.	low	upp	# pack	med.	low	upp	# pack	med.	low	upp	# pack	med.	low	upp	# pack
60	0.01	0.01	0.02	1727	0.06	0.05	0.06	5534	0.38	0.29	0.44	44566	1.02	0.79	1.38	1×10^5	7.39	3.91	8.73	3×10^5	39.30	29.96	56.11	1×10^6	106.04	65.85	144.22	3×10^6
70	0.02	0.01	0.02	1948	0.06	0.05	0.07	6321	0.46	0.37	0.52	53643	1.34	0.97	1.96	1×10^5	8.51	6.39	10.85	4×10^5	53.79	41.79	68.32	2×10^6	158.50	110.38	210.63	5×10^6
80	0.02	0.02	0.02	2210	0.07	0.05	0.07	7294	0.57	0.47	0.64	65301	1.94	1.75	2.31	2×10^5	12.48	7.52	13.51	5×10^5	82.53	69.75	98.70	3×10^6	266.01	224.57	359.33	6×10^6
100	0.02	0.02	0.03	2655	0.07	0.06	0.07	8652	0.80	0.66	0.86	82794	3.03	2.37	3.56	2×10^5	18.54	14.43	23.35	7×10^5	160.79	127.88	178.73	4×10^6	462.48	376.44	517.59	1×10^7
150	0.03	0.02	0.03	3128	0.09	0.08	0.10	10261	0.97	0.78	1.11	1×10^5	5.28	4.17	5.55	3×10^5	26.48	19.24	33.50	9×10^5	215.04	121.05	248.88	5×10^6	679.03	-	-	2×10^7
200	0.02	0.02	0.03	3340	0.10	0.09	0.10	10936	0.99	0.91	1.01	1×10^5	5.30	4.73	5.77	4×10^5	27.39	24.41	29.01	1×10^6	324.04	-	-	7×10^6	1115.65	-	-	2×10^7
300	0.01	0.01	0.02	3432	0.09	0.08	0.10	11330	0.96	0.93	1.01	1×10^5	6.04	5.63	6.96	4×10^5	27.71	27.07	29.18	1×10^6	301.28	-	-	8×10^6	1084.21	-	-	2×10^7
500	0.01	0.01	0.01	3477	0.06	0.05	0.08	11469	0.80	0.77	0.90	1×10^5	5.66	5.15	6.02	4×10^5	27.48	25.90	31.20	1×10^6	356.01	-	-	8×10^6	1109.46	-	-	2×10^7

PACKING EQUIVALENCE CLASSES

	300				400				500				600				700				800				900			
	med.	low	upp	# pack	med.	low	upp	# pack	med.	low	upp	# pack	med.	low	upp	# pack	med.	low	upp	# pack	med.	low	upp	# pack	med.	low	upp	# pack
60	0.00	0.00	0.00	281	0.00	0.00	0.00	524	0.03	0.02	0.03	5555	0.33	0.25	0.38	25853	0.64	0.47	0.81	50518	8.01	4.83	15.86	3×10^5	32.80	17.17	48.84	1×10^6
70	0.00	0.00	0.00	303	0.00	0.00	0.01	569	0.04	0.03	0.04	6327	0.37	0.29	0.45	30505	0.73	0.67	0.86	60849	9.56	8.01	10.34	4×10^5	43.81	25.19	64.88	1×10^6
80	0.00	0.00	0.00	329	0.00	0.00	0.01	625	0.04	0.04	0.04	7299	0.48	0.36	0.56	36765	0.96	0.89	1.10	74909	13.38	10.32	15.22	5×10^5	73.35	45.26	81.95	2×10^6
100	0.00	0.00	0.00	370	0.01	0.01	0.01	699	0.05	0.04	0.06	8644	0.49	0.38	0.61	46854	1.19	1.10	1.80	1×10^5	18.61	15.15	33.61	6×10^5	102.59	86.19	140.68	3×10^6
150	0.00	0.00	0.00	412	0.01	0.01	0.01	786	0.04	0.04	0.05	10237	0.56	0.49	0.62	58950	1.47	0.96	3.07	1×10^5	21.64	15.06	26.16	9×10^5	145.84	106.05	182.35	3×10^6
200	0.00	0.00	0.00	429	0.01	0.01	0.01	821	0.07	0.06	0.08	10904	0.60	0.53	0.79	64416	1.82	1.56	3.59	1×10^5	26.50	-	-	1×10^6	162.30	-	-	5×10^6
300	0.00	0.00	0.00	436	0.01	0.01	0.01	841	0.05	0.04	0.05	11286	0.71	0.59	0.81	67163	2.59	1.64	3.16	1×10^5	304.40	-	-	1×10^6	162.70	-	-	5×10^6
500	0.00	0.00	0.00	440	0.01	0.01	0.01	849	0.09	0.08	0.10	11417	0.73	0.61	0.89	68213	2.81	2.17	4.46	2×10^5	673.20	-	-	1×10^6	154.10	-	-	5×10^6