Theoretical Computer Science 409 (2008) 351–363

Contents lists available at ScienceDirect

Theoretical Computer Science

journal homepage: www.elsevier.com/locate/tcs

Sequential vector packing☆

Mark Cieliebak[b], Alexander Hall[c], Riko Jacob[d], Marc Nunkesser[a,*]

[a] Department of Computer Science, ETH Zurich, Switzerland
[b] sd & m Schweiz AG, 8050 Zurich, Switzerland
[c] Google Switzerland, Brandschenkestr. 110, CH-8002 Zurich, Switzerland
[d] Institut für Informatik der Technische Universität München, Germany

ARTICLE INFO

Article history:
Received 2 October 2007
Received in revised form 11 July 2008
Accepted 23 July 2008
Communicated by J. Díaz

Keywords:
Bin packing
Vector packing
Approximation algorithms
Algorithm design

ABSTRACT

We introduce a novel variant of the well known d-dimensional bin (or vector) packing problem. Given a sequence of non-negative d-dimensional vectors, the goal is to pack these into as few bins as possible, of the smallest possible size. In the classical problem, the bin size vector is given and the sequence can be partitioned arbitrarily. We study a variation where the vectors have to be packed in the order in which they arrive, and the bin size vector can be chosen once in the beginning. This setting gives rise to two combinatorial problems: one in which we want to minimize the number of bins used for a given total bin size, and one in which we want to minimize the total bin size for a given number of bins. We prove that both problems are $\mathcal{NP}$-hard, and propose an LP based bicriteria $(\frac{1}{\varepsilon}, \frac{1}{1-\varepsilon})$-approximation algorithm. We give a 2-approximation algorithm for the version with a bounded number of bins. Furthermore, we investigate properties of natural greedy algorithms, and present an easy to implement heuristic, which is fast and performs well in practice. Experiments with the heuristic and an ILP formulation yield promising results on real world data.

Suppose you want to spray a long text on a wall using stencils for the letters and spray color. You start from the left and assemble as much of the beginning of the text as you have matching stencils at your disposal. Then you mask the area around the stencils and start spraying. Afterwards, you remove the stencils again, so that you can reuse them in the next steps. You iterate this procedure starting after the last letter that was sprayed until the whole text is finished. The sequential unit vector packing problem can be formulated as the following question: if you have bought enough material to produce at most B stencils before you start, how many stencils b_i of each letter $i \in \{A \ldots Z\}$ do you produce in order to minimize the number of steps that you need to spray the whole text?

The problem can be seen as an inverse vector packing problem: the sequence in which the items (characters, interpreted here as unit vectors) occur is fixed and cannot be altered. On the other hand, the bin size vector (here $(b_A, \ldots, b_Z)$) can be changed, as long as its component-wise sum does not exceed a given value B. An equivalent problem was posed to us by an industry partner from the manufacturing industry, where *exactly* this question arises in a production process. As a small letdown we are not allowed to give the details of this process. In this paper we study both this problem and the slightly generalized version where the vectors in the sequence are not restricted to unit vectors. Formally, the sequential vector packing problem is defined as follows:

☆ Work partially supported by European Commission - Fet Open project DELIS IST-001907 Dynamically Evolving Large Scale Information Systems, for which funding in Switzerland is provided by SBF grant 03.0378-1.

* Corresponding author. Tel.: +41 786723448; fax: +41 446321399.
E-mail addresses: mark.cieliebak@sdm.com (M. Cieliebak), alex.hall@gmail.com (A. Hall), jacob@in.tum.de (R. Jacob), marc.nunkesser@inf.ethz.ch (M. Nunkesser).

0304-3975/$ – see front matter © 2008 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2008.07.027

Definition 1 (*Sequential Vector Packing, SVP*).
Given: a sequence $\mathbf{S} = \mathbf{s}_1 \cdots \mathbf{s}_n$ of *demand vectors* $\mathbf{s}_i = (s_{i1}, \ldots, s_{id}) \in \mathbb{Q}_+^d$, $d \in \mathbb{N}$.
Evaluation: for a *bin vector* (or short: *bin*) $\mathbf{b} = (b_1, \ldots, b_d) \in \mathbb{Q}_+^d$ of *total bin size* $B = |\mathbf{b}|_1 = \sum_{j=1}^{d} b_j$, the sequence $\mathbf{s}_1 \cdots \mathbf{s}_n$ can be packed in this order into k bins, if *breakpoints* $0 = \pi_0 < \pi_1 < \cdots < \pi_k = n$ exist, such that

$$\sum_{i=\pi_l+1}^{\pi_{l+1}} \mathbf{s}_i \leq \mathbf{b} \quad \text{for } l \in \{0, \ldots, k-1\} \ ,$$

where the inequalities over vectors are component-wise.
We denote the minimum number of bins for given $\mathbf{S}$ and $\mathbf{b}$ by $\kappa(\mathbf{b}, \mathbf{S}) = k$. This number can be computed in linear time. We denote the jth component, $j \in \{1, \ldots, d\}$, of the demand vectors and the bin vector as *resource* j, i.e., s_{ij} is the demand for resource j of the ith demand vector. We also refer to $\mathbf{s}_i$ as *position* i.
Goals: minimize the total bin size and the number of bins. We formulate this bicriteria objective in the following two versions:

Bounded Size SVP for a given bin size B, find a bin vector $\mathbf{b}$ with $B = |\mathbf{b}|_1$, such that $\kappa(\mathbf{b}, \mathbf{S})$ is a minimum.
Bounded Number SVP for a given number of bins k, find a bin vector $\mathbf{b}$ with $\kappa(\mathbf{b}, \mathbf{S}) = k$, such that the total bin size $B = |\mathbf{b}|_1$ is a minimum.

The *sequential unit vector packing (SUVP)* problem considers the restricted variant where $\mathbf{s}_i$, $i \in \{1, \ldots, n\}$, contains exactly one entry equal to 1; all others are zero. Note that any solution for this version can be transformed in such a way that the bin vector is integral, i.e., $\mathbf{b} \in \mathbb{N}^d$, by potentially rounding down resource amounts to the closest integer (therefore one may also restrict the total bin size to $B \in \mathbb{N}$). The same holds if all vectors in the sequence are integral, i.e., $\mathbf{s}_i \in \mathbb{N}^d$, $i \in \{1, \ldots, n\}$.

Given the bicriteria objective function, it is natural to consider bicriteria approximation algorithms: we call an algorithm a *bicriteria* (α, β)*-approximation algorithm* for the sequential vector packing problem if it finds for each sequence $\mathbf{S}$ and bin size $\beta \cdot B$, a solution which needs no more than α times the number of bins of an optimal solution for $\mathbf{S}$ and bin size B.

Related Work. There is an enormous wealth of publications, both on the classical bin packing problem and on variants of it. The two surveys by Coffman, Garey and Johnson [1,2] give many pointers to the relevant literature until 1997. In [3] Coppersmith and Raghavan introduce the multidimensional (on-line) bin packing problem. There are also some variants that take into consideration precedence relations on the items [17,16] that remotely resemble our setting. Galambos and Woeginger [8] give a comprehensive overview over the on-line bin-packing and vector-packing literature. We like to stress though, that our problem is not an on-line problem, since we are given the complete (albeit immutable) sequence in the beginning. We are unaware of any publication that deals with the sequential vector packing problem. In the context of scheduling algorithms, allowing a certain relaxation in bicriteria approximations (here increasing the bin size) is also called resource augmentation, cf. [10,13].

New Contributions and Outline. In Section 1 we present approximation algorithms for the sequential vector packing problem. These are motivated by the strong $\mathcal{NP}$-hardness results that we give in Section 2. The approximation algorithms are based on an LP relaxation and two different rounding schemes, yielding a bicriteria $(\frac{1}{\varepsilon}, \frac{1}{1-\varepsilon})$-approximation and—as our main result—a 2-approximation for the bounded number version of the problem. Recall that the former algorithm, e.g., for $\varepsilon = \frac{1}{3}$, yields solutions with at most 3 times the optimal number of bins, while using at most 1.5 times the given total bin size B, the latter may use at most the optimal number of bins and at most twice the given total bin size B. In Section 3 we present two simple greedy strategies and argue why they perform badly in the worst case. Furthermore, we give an easy to implement heuristic and present two optimizations concerning subroutines. In particular, we show how to compute $\kappa(\mathbf{b}, \mathbf{S})$ in time $O(\kappa(\mathbf{b}, \mathbf{S}) \cdot d)$ after a preprocessing phase which takes $O(n)$ time. Finally, in Section 4 we discuss the results of experiments with the heuristics and an ILP formulation on real world data.

1. Approximation algorithms

We present an ILP formulation, which we subsequently relax to an LP. We continue by describing a simple rounding scheme which yields a bicriteria $(\frac{1}{\varepsilon}, \frac{1}{1-\varepsilon})$-approximation for bounded size and bounded number SVP and then show how to obtain a 2-approximation for bounded number SVP.

1.1. ILP formulation

For a given sequence $\mathbf{S}$, let $\mathbf{w}_{u,v} := \sum_{i=u+1}^{v} \mathbf{s}_i$, for $u, v \in \{0, \ldots, n\}$ and $u < v$, denote the *total demand* (or *total demand vector*) of the subsequence $\mathbf{S}_{u,v} := \mathbf{s}_{u+1} \cdots \mathbf{s}_v$. If $\mathbf{w}_{u,v} \leq \mathbf{b}$ holds, we can pack the subsequence $\mathbf{S}_{u,v}$ into bin $\mathbf{b}$.

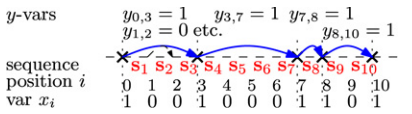


Fig. 1. An exemplary instance, its potential ILP solution, and its flow representation. The solution needs 4 bins.

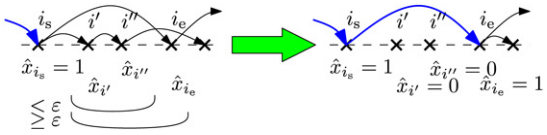


Fig. 2. An example of the rerouting of flow in Lines 4 (a)-(c) of the algorithm. Assume that the flow splits at i_s into a flow of value 0.8 (top) and one of value 0.2 (bottom), and $\varepsilon = 0.6$. In step (a) $\hat{y}_{i_s,i_e}$ is set to 1, in (b) the new edge starting from i_e is created with flow value 0.2, in (c) the three short bottom edges are deleted. The resulting flow is a valid flow again.

The following integer linear programming (ILP) formulation solves both versions of the sequential vector packing problem. Let $X := \{x_i | i \in \{0, \ldots, n\}\}$ and $Y := \{y_{u,v} | u, v \in \{0, \ldots, n\}, u < v\}$ be two sets of 0-1 variables.

$$\text{minimize} \quad k \text{ (or } B\text{)}$$

$$\text{s.t.} \quad x_0 = 1 \tag{1}$$

$$\sum_{u=0}^{i-1} y_{u,i} = \sum_{v=i+1}^{n} y_{i,v} = x_i \quad \text{for } i \in \{1, \ldots, n-1\} \tag{2}$$

$$\sum_{\substack{u,v:\\ u<i\leq v}} \mathbf{w}_{u,v} \cdot y_{u,v} \leq \mathbf{b} \quad \text{for } i \in \{1, \ldots, n\} \tag{3}$$

$$\sum_{j=1}^{d} b_j = B \tag{4}$$

$$\sum_{i=1}^{n} x_i = k \tag{5}$$

$$B \in \mathbb{Q}_+, \text{ (or } k \in \mathbb{N}\text{)}, \quad \mathbf{b} \in \mathbb{Q}_+^d, x_i, y_{u,v} \in \{0, 1\} \quad \text{for } x_i \in X, y_{u,v} \in Y$$

The 0-1 variable x_i indicates whether there is a breakpoint at position $i \geq 1$. The 0-1 variable $y_{u,v}$ can be seen as a flow which is routed on an edge from position $u \in \{0, \ldots, n-1\}$ to position $v \in \{1, \ldots, n\}$, with $u < v$. In an integral solution a unit flow goes from breakpoint to breakpoint along edges (u, v) with $y_{u,v} = x_u = x_v = 1$, see Fig. 1. The Constraints (2) ensure that flow conservation holds for the flow represented by the $y_{u,v}$ variables and that x_i is equal to the inflow (outflow) which enters (leaves) position i. Constraint (1) enforces that only one unit of flow is sent via the Y variables. The path which is taken by this unit of flow directly corresponds to a series of breakpoints.

In Constraints (3) the bin vector $\mathbf{b}$ comes into play: for any two consecutive breakpoints (e.g., $x_u = x_v = 1$) the constraint ensures that the bin vector is large enough for the total demand between the breakpoints (e.g., the total demand $\mathbf{w}_{u,v}$ of the subsequence $\mathbf{S}_{u,v}$). Note that Constraints (3) sum over all edges that span over a position i (in a sense the cut defined by position i), enforcing that the total resource usage is bounded by $\mathbf{b}$. For the two consecutive breakpoints x_u and x_v, this amounts to $\mathbf{w}_{u,v} \cdot y_{u,v} \leq \mathbf{b}$. Finally, Constraints (4) and (5) ensure the correct total size of the bin vector and the correct number of bins.

1.2. An easy $(\frac{1}{\varepsilon}, \frac{1}{1-\varepsilon})$-approximation

As a first step, we relax the ILP formulation to an LP: here this means to have $x_i, y_{u,v} \in [0, 1]$ for $x_i \in X, y_{u,v} \in Y$. The following Algorithm Eps Rounding computes a $(\frac{1}{\varepsilon}, \frac{1}{1-\varepsilon})$-approximation:

1. Solve the LP optimally with one of the two objective functions. Let $(X^\star, Y^\star, \mathbf{b}^\star)$ be the obtained fractional solution.
2. Set $(\hat{X}, \hat{Y}, \hat{\mathbf{b}}) = (X^\star, Y^\star, \frac{1}{1-\varepsilon} \cdot \mathbf{b}^\star)$ and $i_s = 0$. Stepwise round $(\hat{X}, \hat{Y})$:
3. Let $i_e > i_s$ be the first position for which $\sum_{i=i_s+1}^{i_e} \hat{x}_i \geq \varepsilon$.
4. Set $\hat{x}_i = 0$ for $i \in \{i_s + 1, \ldots, i_e - 1\}$, set $\hat{x}_{i_e} = 1$. Reroute the flow accordingly (see also Fig. 2): (a) Set $\hat{y}_{i_s,i_e} = 1$. (b) Increase $\hat{y}_{i_e,i}$ by $\sum_{i'=i_s}^{i_e-1} \hat{y}_{i',i}$, for $i > i_e$. (c) Set $\hat{y}_{i_s,i'} = 0$ and $\hat{y}_{i',i} = 0$, for $i' \in \{i_s + 1, \ldots, i_e - 1\}, i > i'$.
5. Set the new i_s to i_e and continue in Line 3, until $i_s = n$.

Theorem 1. *The algorithm* Eps Rounding *is a* $(\frac{1}{\varepsilon}, \frac{1}{1-\varepsilon})$*-approximation algorithm for the sequential vector packing problem.*

Proof. We show the desired result in three separate steps.

Integer Rounding. The following invariant is easy to see, by considering Fig. 2: at the beginning of each iteration step (i.e., at Line 3) the current, partially rounded solution $(\hat{X}, \hat{Y}, \hat{\mathbf{b}})$ corresponds to a valid flow, which is integral until position i_s. From this invariant it follows that $(\hat{X}, \hat{Y}, \hat{\mathbf{b}})$ in the end corresponds to a valid integer flow.

At Most $\frac{1}{\varepsilon}$-times the Number of Breakpoints. In Line 4, $\hat{x}_i$ values which sum up to at least ε (see Line 3) are replaced by $\hat{x}_{i_e} = 1$. Therefore, the rounding increases the total value of the objective function by at most a factor of $\frac{1}{\varepsilon}$.

At Most $\frac{1}{1-\varepsilon}$-times the Total Bin Size. Again consider one step of the iteration. We need to check that by rerouting the flow to go directly from i_s to i_e we do not exceed the LP bin capacity by more than $\frac{1}{1-\varepsilon} \cdot \mathbf{b}^\star$. We show the stronger invariant that in each step after rerouting, the current, partially rounded solution fulfills Constraint (3) until i_e w.r.t. $\hat{\mathbf{b}}$ and from $i_e + 1$ to the end of the sequence w.r.t. $\mathbf{b}^\star$. First let us consider the increase of $\hat{y}_{i_e,i}$, for $i > i_e$, in Line 4 (b). Since the total increase is given directly by some $\hat{y}_{i',i}$ which are set to 0 (Line 4 (c)), the Constraint (3) still holds after the change w.r.t. $\mathbf{b}^\star$. In other words, the flow on edges is rerouted here onto shorter (completely contained) edges; this does not change the feasibility of Constraint (3), since the only difference now is that for some i the sum in (3) is over fewer terms.

Now we consider the increase of $\hat{y}_{i_s,i_e}$ to 1. We show that the total demand w_{i_s,i_e} between the new breakpoints i_s and i_e is bounded by $\frac{1}{1-\varepsilon} \cdot \mathbf{b}^\star$. With $\hat{x}_{i_s} = 1$ and since $\sum_{i=i_s+1}^{i_e-1} \hat{x}_i < \varepsilon$ (see Line 3), we know that $\sum_{i=i_e}^{n} \hat{y}_{i_s,i} = \hat{x}_{i_s} - \sum_{i=i_s+1}^{i_e-1} \hat{y}_{i_s,i} \geq 1 - \sum_{i=i_s+1}^{i_e-1} \hat{x}_i > 1 - \varepsilon$; note that the first equality holds by the flow conservation constraint (2). For w_{i_s,i_e} we obtain $w_{i_s,i_e} \cdot \sum_{i=i_e}^{n} \hat{y}_{i_s,i} \leq \sum_{i=i_e}^{n} w_{i_s,i} \cdot \hat{y}_{i_s,i} \leq \sum_{u,v:u<i_e\leq v} w_{u,v} \cdot \hat{y}_{u,v} \leq \mathbf{b}^\star$, where the last inequality follows by the invariant for the last step. Thus, plugging these two inequalities together, we know for the total demand $w_{i_s,i_e} < \frac{1}{1-\varepsilon} \cdot \mathbf{b}^\star = \hat{\mathbf{b}}$. Since this holds for all iteration steps and thus for all consecutive breakpoints of the final solution, it is clear that multiplying the bin vector of the LP solution by a factor $\frac{1}{1-\varepsilon}$ yields a valid solution for the ILP. □

Note that one would not actually implement the algorithm Eps Rounding. Instead, it suffices to compute the bin vector $\mathbf{b}^\star$ with the LP and then multiply it by $\frac{1}{1-\varepsilon}$ and evaluate the obtained bin vector, e.g., with the algorithm given in Section 3.

1.3. A 2-approximation for bounded number Sequential Vector Packing

We start by proving some properties of the LP relaxation and then describe how they can be applied to obtain the rounding scheme, yielding the desired approximation ratio.

1.3.1. Properties of the relaxation

Let $(X, Y, \mathbf{b})$ be a fractional LP solution w.r.t. one of the objective functions; recall that the Y variables represent a flow. Let $e_1 = (u, v)$ and $e_2 = (u', v')$ denote two flow carrying edges, i.e., $y_{u,v} > 0$ and $y_{u',v'} > 0$. We say that e_1 *is contained in* e_2 if $u' < u$ and $v' > v$, we also call (e_1, e_2) an *embracing pair*. We say an embracing pair (e_1, e_2) is *smaller* than an embracing pair $(\hat{e}_1, \hat{e}_2)$, if the length of e_1 (for $e_1 = (u, v)$, its length is $v - u$) is less than the length of $\hat{e}_1$ and in case of equal lengths, if $u < \hat{u}$ (order by left endpoint). That is, for two embracing pairs with distinct e_1 and $\hat{e}_1$ we always have that one is smaller than the other. We show that the following structural property holds:

Lemma 1 (*No Embracing Pairs*). *Any optimal fractional LP solution* $(X^\star, Y^\star, \mathbf{b}^\star)$ *can be modified in such a way that it contains no embracing pairs, without increasing the number of bins and without modifying the bin vector.*

Proof. We set $Y = Y^\star$ and show how to stepwise treat embracing pairs contained in Y, proving after each step that $(X^\star, Y, \mathbf{b}^\star)$ is still a feasible LP solution. We furthermore show that this procedure terminates, and in the end no embracing pairs are left in Y.

Let us begin by describing one iteration step, assuming $(X^\star, Y, \mathbf{b}^\star)$ to be a feasible LP solution which still contains embracing pairs. Let (e_1, e_2), with $e_1 = (u, v)$ and $e_2 = (u', v')$, be an embracing pair. We now modify the flow Y to obtain a new flow Y' by rerouting $\lambda = \min\{y_{u,v}, y_{u',v'}\}$ units of flow from e_1, e_2 onto the edges $e'_1 = (u, v')$ and $e'_2 = (u', v)$: $y'_{u,v} = y_{u,v} - \lambda$, $y'_{u',v'} = y_{u',v'} - \lambda$ and $y'_{u',v} = y_{u',v} + \lambda$, $y'_{u,v'} = y_{u,v'} + \lambda$; see also Fig. 3 (left). The remaining flow values in Y' are taken directly from Y. It is easy to see that the flow conservation constraints (2) still hold for the values $X^\star$, Y' (consider a circular flow of λ units sent in the residual network of Y on the cycle u', v, u, v', u'). Since $X^\star$ is unchanged this also implies that the number of bins did not change, as desired. It remains to prove that the Constraints (3) still hold for the values Y', $\mathbf{b}^\star$ and to detail how to consecutively choose embracing pairs (e_1, e_2) in such a way that the iteration terminates.

Feasibility of the Modified Solution. Constraints (3) are parameterized over $i \in \{1, \ldots, n\}$. We argue that they are not violated separately for $i \in \{u' + 1, \ldots, u\}$, $i \in \{u + 1, \ldots, v\}$, and $i \in \{v + 1, \ldots, v'\}$, i.e., the regions b, c, and d in Fig. 3 (left). For the remaining regions a and e it is easy to check that the values of the affected variables do not change when replacing Y by Y'. So let us consider the three regions:

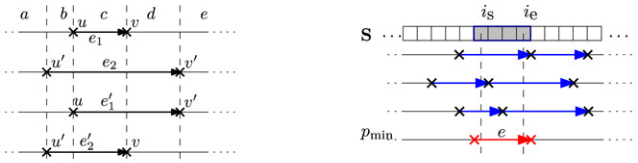


Fig. 3. Left: Replacement of λ units of flow on e_1 and e_2 by λ units of flow on e'_1 and e'_2 in Lemma 1. Right: Extracting the integral solution. Edge e together with other potential edges in $Y^\star$ in Theorem 2.

Region b (d). The only variables in (3) which change when replacing Y by Y' for this region are: $y'_{u',v'} = y_{u',v'} - \lambda$ and $y'_{u',v} = y_{u',v} + \lambda$. This means that flow is moved to a shorter edge, which can only increase the slack of the constraints: With $\mathbf{w}_{u',v} < \mathbf{w}_{u',v'}$ it is easy to see that (3) still holds in region b. Region d is analogous to b.

Region c. Here the only variables which change in (3) are: $y'_{u,v} = y_{u,v} - \lambda$, $y'_{u',v'} = y_{u',v'} - \lambda$, $y'_{u',v} = y_{u',v} + \lambda$, and $y'_{u,v'} = y_{u,v'} + \lambda$. In other words, λ units of flow were moved from e_1 to e'_1 and from e_2 to e'_2. Let us consider the fraction of demand which is contributed to (3) by these units of flow before and after the modification. Before (on e_1 and e_2) this was $\lambda \cdot (\mathbf{w}_{u,v} + \mathbf{w}_{u',v'})$ and afterwards (on e'_1 and e'_2) it is $\lambda \cdot (\mathbf{w}_{u',v} + \mathbf{w}_{u,v'})$. Since both quantities are equal, the left hand side of (3) remains unchanged in region c.

Choice of (e_1, e_2) and Termination of the Iteration. In each step of the iteration, we always choose the smallest embracing pair (e_1, e_2), as defined above. If there are several smallest embracing pairs (which by definition all contain the same edge e_1), we choose one of these arbitrarily.

First we show that the modification does not introduce an embracing pair which is smaller than (e_1, e_2). We assume the contrary and say w.l.o.g. that the flow added to edge e'_1 creates a new embracing pair (e, e'_1) which is smaller than the (removed) embracing pair (e_1, e_2). Clearly, e is also contained in e_2. Therefore, before the modification (e, e_2) would have been an embracing pair as well. Since (e, e_2) is smaller than (e_1, e_2) it would have been chosen instead, which gives the contradiction.

It follows that we can divide the iterations into a bounded number of phases: in each phase all considered embracing pairs are with respect to the same e_1-type edge. As soon as a phase is finished (i.e., no embracing pairs with the phase's e_1-type edge remain) this e_1-type edge will never be considered again, since this could only happen by introducing a smaller embracing pair later in the iteration. Thus, there are at most $O(n^2)$ phases.

Now we consider a single phase, during which an edge e_1 is contained in possibly several other edges e_2. By the construction of the modification for an embracing pair (e_1, e_2), it is clear that e_2 could not be chosen twice in the same phase. Therefore, the number of modification steps per phase can also be bounded by $O(n^2)$. □

1.3.2. Choose a flow carrying path

We will use the structural insights from above to prove that bin vector $2 \cdot \mathbf{b}^\star$ yields a 2-approximation for bounded number SVP.

Due to Lemma 1 an optimal fractional LP solution $(X^\star, Y^\star, \mathbf{b}^\star)$ without embracing pairs exists. Let $p_{\min}$ denote the shortest flow carrying path in $(X^\star, Y^\star, \mathbf{b}^\star)$, where shortest is meant with respect to the number of breakpoints. Clearly, the length of $p_{\min}$ is at most the number of bins $\sum_{i=1}^{n} x_i^\star$, since the latter can be seen as a linear combination of the path lengths of an arbitrary path decomposition. Below, we show that the integral solution corresponding to $p_{\min}$ is feasible for the bin vector $2 \cdot \mathbf{b}^\star$, and thus $p_{\min}$ and $2 \cdot \mathbf{b}^\star$ give a 2-approximation.

Observe that the approximation algorithm does not actually need to transform an optimal LP solution, given, e.g., by an LP solver, into a solution without embracing pairs. The existence of path $p_{\min}$ in such a transformed solution is merely taken as a proof that the bin vector $2 \cdot \mathbf{b}^\star$ yields less than $\sum_{i=1}^{n} x_i^\star$ breakpoints. To obtain such a path, we simply evaluate $2 \cdot \mathbf{b}^\star$ with the trivial algorithm without preprocessing discussed in Section 3 ($\mathbf{b}^\star$ given by the LP solver).

Theorem 2. *Given an optimal fractional LP solution $(X^\star, Y^\star, \mathbf{b}^\star)$ without embracing pairs, let $p_{\min}$ denote the shortest flow carrying path. The integral solution corresponding to $p_{\min}$ is feasible for $2 \cdot \mathbf{b}^\star$.*

Proof. We only have to argue for the feasibility of the solution w.r.t. the doubled bin vector. Again, we will consider Constraints (3). Fig. 3 (right) depicts an edge e on path $p_{\min}$ and other flow carrying edges. We look at the start and end position i_s and i_e in the subsequence defined by e. Denote by $E_{i_s} = \{(u, v) | 0 \le u < i_s \le v \le n\}$ (and E_{i_e}, respectively) the set of all flow carrying edges that cross i_s (i_e) and by $i_{\min}$, ($i_{\max}$) the earliest tail (latest head) of an arc in E_{i_s}, (E_{i_e}). Furthermore, let $E' = E_{i_s} \cup E_{i_e}$. Summing up the two Constraints (3) for i_s and i_e gives $2\mathbf{b}^\star \ge \sum_{(u,v)\in E_{i_s}} y^\star_{u,v} \cdot \mathbf{w}_{u,v} + \sum_{(u,v)\in E_{i_e}} y^\star_{u,v} \cdot \mathbf{w}_{u,v} =: A$

and thus

$$2\mathbf{b}^\star \geq A \geq \sum_{i_{\min}<i\leq i_{\max}} \sum_{\substack{(u,v)\in E':\\ u<i\leq v}} y^\star_{u,v}\cdot \mathbf{s}_i \tag{6}$$

$$\geq \sum_{i_s<i\leq i_e} \sum_{\substack{(u,v)\in E':\\ u<i\leq v}} y^\star_{u,v}\cdot \mathbf{s}_i = \sum_{i_s<i\leq i_e} \mathbf{s}_i = \mathbf{w}_{i_s,i_e} \ . \tag{7}$$

The second inequality in (6) is in general an inequality, because the sets E_{i_s} and E_{i_e} need not be disjoint. For the first equality in (7) we rely on the fact that there are no embracing pairs. For this reason, each position between i_s and i_e is covered by an edge that covers either i_s or i_e. In particular, no other edge in $E \setminus E'$ intersecting position i can carry flow so that the sum $\sum_{\substack{(u,v)\in E':\\ u<i\leq v}} y^\star_{u,v}$ must evaluate to one for $i_s < i \leq i_e$. We have shown that the demand between any two breakpoints on $p_{\min}$ can be satisfied by the bin vector $2\cdot\mathbf{b}^\star$. □

Observe that for integral resources, the above proof implies that even $\lfloor 2\mathbf{b}^\star\rfloor$ has no more breakpoints than the optimal solution. Therefore, the whole algorithm boils down to computing $\lfloor 2\mathbf{b}^\star\rfloor$. Note also that it is easy to adapt both approximation algorithms so that they can handle pre-specified breakpoints. The corresponding x_i values can simply be set to one in the ILP and LP formulations.

2. NP-hardness

For all considered problem variants it is easy to determine the objective value once a bin vector is chosen. Hence, for all variants of the sequential vector packing problem considered in this article, the corresponding decision problem is in $\mathcal{NP}$. Moreover, the decision problem of both the bounded size and bounded number versions are identical. Therefore, we will not distinguish between the two versions here. We now come to the $\mathcal{NP}$-hardness result. To simplify the exposition, we first consider a variant of the sequential unit vector packing problem where the sequence of vectors has prespecified breakpoints, which are precisely w positions apart. Then the sequence effectively decomposes into a set of *windows* of length w, and for each position in such a window i it is sufficient to specify the resource that is used at position $h \in \{1,\ldots,w\}$, denoted as $r^i_h \in \{1,\ldots,d\}$. This situation can be understood as a set of sequential unit vector packing problems that have to be solved with the same bin vector. The objective is to minimize the total number of (additional) breakpoints, i.e., the sum of the objective functions of the individual problems. Then, we also show strong $\mathcal{NP}$-hardness for the original problem.

Lemma 2. *Finding the optimal solution for sequential unit vector packing with windows of length* 4 *(dimension d and bin size B as part of the input) is $\mathcal{NP}$-hard.*

Proof. By reduction from the $\mathcal{NP}$-complete problem k-densest subgraph [7], which generalizes the well known problem Clique [9]. Let $G = (V, E)$ be an instance of k-densest subgraph, i.e., an undirected graph without isolated vertices in which we search for a subset of vertices of cardinality k that induces a subgraph with the maximal number of edges.

We construct a sequential unit vector packing instance $(\mathbf{S}, B)$ with windows of length 4 and with $d = |V|$ resources. Assume as a naming convention $V = \{1,\ldots,d\}$. There is precisely one window per edge $e = (u, v) \in E$; the sequence of this window is $s^e = uvuv$. The total bin size is set to $B = d + k$. This transformation can be carried out in polynomial time and achieves, as shown in the following, that $(\mathbf{S}, B)$ can be solved with at most $|E| - \ell$ (additional) breakpoints if and only if G has a subgraph with k vertices containing at least ℓ edges.

Because every window contains at most two vectors of the same resource, having more than two units of one resource does not influence the number of breakpoints. Every resource has to be assigned at least one unit because there are no isolated vertices in G. Hence, a solution to $(\mathbf{S}, B)$ is characterized by the subset R of resources to which two units are assigned (instead of one). By the choice of the total bin size we have $|R| = k$. A window does not induce a breakpoint if and only if both its resources are in R, otherwise it induces one breakpoint.

If G has a node induced subgraph G' of size k containing ℓ edges, we choose R to contain the vertices of G'. Then, every window corresponding to an edge of G' has no breakpoint, whereas all other windows have one. Hence, the number of (additional) breakpoints is $|E| - \ell$.

If $(\mathbf{S}, B)$ can be scheduled with at most $|E|-\ell$ breakpoints, define R as the resources for which there is more than one unit in the bin vector. Now $|R| \leq k$, and we can assume $|R| = k$, since the number of breakpoints only decreases if we change some resource from one to two, and changing the number of resources from something larger than two to two does not increase the number of breakpoints. The set R defines a subgraph G' with k vertices of G. The number of edges is at least ℓ because only windows with both resources in R do not have a breakpoint. □

It remains to consider the original problem without pre-specified breakpoints.

Lemma 3. *Let $(\mathbf{S}, B)$ be an instance of sequential (unit) vector packing of length n with k pre-specified breakpoints and d resources ($d \leq B$) where every resource is used at least once. Then one can construct in polynomial time an instance $(\mathbf{S}', B')$ of the (unit) vector packing problem with bin size $B' = 3B+2$ and $d' = d+2B+2$ resources that can be solved with at most $\ell + k$ breakpoints if and only if $(\mathbf{S}, B)$ can be solved with at most ℓ breakpoints.*

Proof. The general idea is to use for every prespecified breakpoint some "stopping" sequence F_i with the additional resources in a way that the bound B' guarantees that there is precisely one breakpoint in F_i. This sequence F_i needs to enforce exactly one breakpoint, no matter whether or not there was a breakpoint within the previous window (i.e., between F_{i-1} and F_i). If we used the same sequence for F_{i-1} and F_i, a breakpoint within the window would yield a "fresh" bin vector for F_i. Therefore, the number of breakpoints in F_i could vary depending on the demands in the window (and whether or not they incur a breakpoint).

To avoid this, we introduce two different stopping sequences F and G which we use alternatingly. This way we are sure that between two occurrences of F there is at least one breakpoint. The resources $1, \ldots, d$ of $(\mathbf{S}', B')$ are one-to-one the resources of $(\mathbf{S}, B)$. The $2B + 2$ additional resources are divided into two groups $f_1, \ldots, f_{B+1}$ for F and $g_1, \ldots, g_{B+1}$ for G. Every odd pre-specified breakpoint in $\mathbf{S}$ is replaced by the sequence $F := f_1 f_2 \cdots f_{B+1} f_1 f_2 \cdots f_{B+1}$ and all even breakpoints by the sequence $G := g_1 g_2 \cdots g_{B+1} g_1 g_2 \cdots g_{B+1}$.

To see the backward direction of the statement in the lemma, a bin $\mathbf{b}$ for $(\mathbf{S}, B)$ resulting in ℓ breakpoints can be augmented to a bin vector $\mathbf{b}'$ for $(\mathbf{S}', B')$ by adding one unit for each of the new resources. This does not exceed the bound B'. Now, in $(\mathbf{S}', B')$ there will be the original breakpoints and a breakpoint in the middle of each inserted sequence. This shows that $\mathbf{b}'$ results in $\ell + k$ breakpoints for $(\mathbf{S}', B')$, as claimed.

To consider the forward direction, let $\mathbf{b}'$ be a solution to $(\mathbf{S}', B')$. Because every resource must be available at least once, and $B' - d' = 3B + 2 - (d + 2B + 2) = B - d$, at most $B - d < B$ entries of $\mathbf{b}'$ can be more than one. Therefore, at least one of the resources f_i is available only once, and at least one of the resources g_j is available only once. Hence, there must be at least one breakpoint within each of the k inserted stopping sequences. Let $k + \ell$ be the number of breakpoints induced by $\mathbf{b}'$ and $\mathbf{b}$ the projection of $\mathbf{b}'$ to the original resources. Since all resources must have at least one unit and by choice of B' and d' we know that $\mathbf{b}$ sums to less than B.

Now, if a subsequence of $\mathbf{S}$ not containing any f or g resources can be packed with the resources $\mathbf{b}'$, this subsequence can also be packed with $\mathbf{b}$. Hence, $\mathbf{b}$ does not induce more than ℓ breakpoints in the instance $(\mathbf{S}, B)$ with pre-specified breakpoints. □

Theorem 3. *The sequential unit vector packing problem is strongly $\mathcal{NP}$-hard.*

Proof. By Lemma 2 and Lemma 3, with the additional observation that all numbers used are polynomial in the size of the original graph. □

Additionally, one can consider the influence of parameters on the complexity of the problem. We say that an algorithm is fixed parameter tractable (FPT) with respect to parameter k if its running time is bounded by $f(k) \cdot n^{O(1)}$ for an arbitrary function f (usually something like 2^k or 2^{2^k}). See for example [11] or [5] for a thorough introduction to the concept.

If the windows in the vector packing problem are limited to length three, the problem can be solved in polynomial time by algorithm Greedy-Grow of Section 3.1: For a window of length three, it is impossible that two resources appear twice in it. More precisely, for a single window the situation is the following: if all resources are different, these three resources are necessary, and there will not be a breakpoint. For a window uvv, vuv, or vvu there is no breakpoint if two v-resources are available, otherwise there will be precisely one breakpoint, i.e., the number of breakpoints is two minus the number of v-resources. For a window of the form vvv, the number of breakpoints is three minus the number of v-resources. Summarizing, there is no interaction between the resources, and the Algorithm Greedy-Grow solves the problem optimally. Additionally, Lemma 2 shows that the problem is $\mathcal{NP}$-hard even if all windows have length 4, where the sequence $s^e = uvuv$ creates a dependency between double availability of resources. Hence, the parameter window size does not allow an FPT-algorithm if $\mathcal{P} \neq \mathcal{NP}$.

On the other hand, for integral $\mathbf{S}$ and B, the parameter B allows (partly because $d \leq B$) to enumerate and evaluate (Sections 3.2.1 and 3.2.2) the number of breakpoints in time $f(B) \cdot n^{O(1)}$, i.e., this is an FPT-algorithm. A constant upper limit on the number of breakpoints allows to enumerate all positions of breakpoints and to determine the necessary bin vector in polynomial time. Note that this is not an FPT algorithm.

3. Practical algorithms

Both the problem presented in the introduction and the original industry problem are bounded size SUVP problems. For this reason, we focus on this variant when considering practical algorithms.

3.1. Greedy algorithms

We describe two natural greedy heuristics for sequential unit vector packing. Recall that we denote by $\kappa(\mathbf{b}, \mathbf{S})$, the minimal number of breakpoints needed for a fixed bin vector $\mathbf{b}$ and given $(\mathbf{S}, B)$. Observe that it is relatively easy to calculate $\kappa(\mathbf{b}, \mathbf{S})$ in linear time (see end of this section and Section 3.2.2). The two greedy algorithms we discuss here are: Greedy-Grow and Greedy-Shrink. Greedy-Grow grows the bin vector starting with the all one vector. In each step it increases some resource by an amount of 1 until the total bin size B is reached, greedily choosing the resource whose increment improves $\kappa(\mathbf{b}, \mathbf{S})$ the most. Greedy-Shrink shrinks the bin vector starting with a bin vector $\mathbf{b} = \sum_{i=1}^{n} \mathbf{s}_i$, which yields $\kappa(\mathbf{b}, \mathbf{S}) = 0$

but initially ignores the bin size B. In each step it then decreases some resource by an amount of 1 until the total bin size B is reached, greedily choosing the resource whose decrement worsens $\kappa(\mathbf{b}, \mathbf{S})$ the least.

In light of the following observations, it is important to specify the tie-breaking rule for the case that there is no improvement at all after the addition of a resource. Greedy-Grow can be forced to produce a solution only by this tie-breaking rule, which is an indicator for its bad performance.

Observation 1. *Given any instance $(\mathbf{S}, B)$ to bounded size SVP, this instance can be modified to an instance $(\mathbf{S}', B')$, with $n' = n, d' = 2d, B' = 2B$ such that all of* Greedy-Grow*'s choices of which resource to add depend entirely on the tie-breaking rule.*

Proof. The idea is to split each resource r into two resources r_1, r_2 and to replace each occurrence of r in a demand vector $\mathbf{s}$ by a demand for r_1 and r_2. We call this transformation *doubling*. Then, considering Greedy-Grow's approach to reduce the number of breakpoints, increasing r_1 or r_2 alone is not enough. Only if r_1 and r_2 are both increased, the number of breakpoints may decrease. That is, for all resources the number of saved breakpoints in the beginning is zero and greedy is forced to take an arbitrary resource in Step 1 and then the partner of this resource in Step 2. Then Greedy-Grow again chooses an arbitrary resource in Step 3 and its partner in Step 4, and so on. □

It follows that Greedy-Grow with an unspecified tie-breaking rule can be led to produce arbitrarily bad solutions. Consider for example the sequence $(uvw)^k = uvwuvwuvw \ldots uvw$. The optimal use of B resources is an equal split, leading to $\lceil \frac{3k}{B} \rceil - 1$ breakpoints. Assume as a tie-breaking rule to always take more of resource u. Now, Greedy-Grow will consider $\mathbf{b} = (i, 1, 1)$, and find that all of $\mathbf{b} = (i, 1, 2)$, $\mathbf{b} = (i, 2, 1)$, or $\mathbf{b} = (i+1, 1, 1)$ produce the same number $k-1$ breakpoints, hence the tie-breaking rule will chose the latter. Hence, the solution of Greedy-Grow has the same number of breakpoints as the solution $(1, 1, 1)$, and is roughly a factor $B/3$ away from the optimum.

Also Greedy-Shrink can produce bad solutions depending on the tie-breaking scheme as the following observation shows.

Observation 2. *There are instances with d resources on which the solution produced by* Greedy-Shrink *is a factor of $\lfloor d/2 \rfloor$ worse than the optimal solution, if the tie-breaking rule can be chosen by the adversary.*

Proof. Let $k = \lfloor d/2 \rfloor$, consider the following unit vector instance with $2k$ resources and $B = 3k$: "$1 \cdots k\ 1 \cdots k(k+1)(k+1)(k+2)(k+2) \cdots (2k)(2k)$". At the beginning of the algorithm $\mathbf{b}$ is set to $(2, \ldots, 2)$. In the first step the removal of each of the resources incurs one breakpoint. Therefore, Greedy-Shrink deletes an arbitrary resource depending on the tie-breaking scheme. We let this resource be one of the last k ones. After this deletion the situation remains unchanged except for the fact that the chosen resource must not be decreased any more. It follows that in k steps Greedy-Shrink sets the last k resources to one, which incurs a total cost of k, whereas the optimal solution sets the first k resources to one, which incurs a cost of 1. Thus, the ratio of greedy versus optimal solution is $k = \lfloor d/2 \rfloor$. □

For the experiments we use for both heuristics a *round-robin* tie-breaking rule that cycles through the resources. Every time a tie occurs, it chooses the cyclic successor of the resource that was increased (decreased) in the last tie. Clearly, round-robin is not necessarily a particularly good tie-breaking rule. We chose it for its simplicity, and because it eventually tries all resources.

3.2. Enumeration heuristic

We present an enumeration heuristic for integral vectors $\mathbf{s}_i \in \mathbb{N}^d, i \in \{1, \ldots, n\}$, that is inspired by a variant of Schöning's 3-SAT algorithm [15] that searches the complete hamming balls of radius $\lfloor n/4 \rfloor$ around randomly chosen assignments, see [4]. The following algorithm uses a similar combination of randomized guessing, and complete enumeration of parts of the solution space that are exponentially smaller than the whole solution space. The idea is to guess uniformly at random (u.a.r.) subsequences $\mathbf{S}_{i_1,i_2}$ of the sequence that do not incur a breakpoint in a fixed optimal solution $\mathbf{b}_{\text{opt}}$. For such a subsequence, we know that $\mathbf{b}_{\text{opt}} \geq \mathbf{w}_{i_1,i_2}$. In particular, if we know a whole set W of such total demand vectors that all come from subsequences without breakpoints for $\mathbf{b}_{\text{opt}}$, we know that $\mathbf{b}_{\text{opt}} \geq \max_{\mathbf{w} \in W} \mathbf{w}$ must hold for a component-wise maximum. This idea leads to the Randomized Heuristic Enumeration (RHE) algorithm:

Phase 1*:* Start with a "lower bound vector" $\mathbf{t} = 0$. For a given subsequence length ssl and a number p of repetitions, in each of p rounds choose $\underline{\sigma}_i =_{\text{u.a.r}} \{0, \ldots, n - \text{ssl}\}$, set $\overline{\sigma}_i = \underline{\sigma}_i + \text{ssl}$, and then set $\mathbf{t} = \max\{\mathbf{t}, \mathbf{w}_{\underline{\sigma}_i, \overline{\sigma}_i}\}$.

Phase 2*:* Find a bin vector $\mathbf{b}$ of total size B with $\mathbf{b} \geq \mathbf{t}$ that minimizes $\kappa(\mathbf{b}, \mathbf{S})$. Do this by enumerating all $\mathbf{b} \geq \mathbf{t}$ of total size B.

As easy as the enumeration in Phase 2 seems, this should be done efficiently. We discuss this aspect in some more detail in the following subsection. It is straightforward to analyze the success probability of this algorithm if we relate the subsequence length to an estimate k' of the minimum number of breakpoints k.

Lemma 4. *Let $\mathbf{b}_{\text{opt}}$ be an optimal bin vector for an integral instance $(\mathbf{S}, B)$, choose ssl as $\lfloor \frac{n}{4k'} \rfloor$ and assume $k' \geq k/2$, where k is the minimal number of breakpoints. Then for each of the demand vectors $\mathbf{w}_{\underline{\sigma}_i, \overline{\sigma}_i}$, $i \in \{1, \ldots, p\}$ it holds that $\Pr[\mathbf{w}_{\underline{\sigma}_i, \overline{\sigma}_i} \leq \mathbf{b}_{\text{opt}}] \geq \frac{1}{2}$.*

Proof. A sufficient (but not necessary) condition for $\mathbf{w}_{\underline{\sigma}_i,\overline{\sigma}_i} \leq \mathbf{b}_{\text{opt}}$, is that the optimal solution $\mathbf{b}_{\text{opt}}$ has no breakpoint in the subsequence $\mathbf{S}_{\underline{\sigma}_i,\overline{\sigma}_i}$. As there are at most $2k'$ such breakpoints the probability that we hit one with a random interval of length $\lfloor \frac{n}{4k'} \rfloor$ is bounded by $\frac{1}{2}$. □

Observe that the first subsequence that is guessed increases the lower bound vector by its full length. Subsequent guesses can, but need not, improve the lower bounds. The growth of the lower bound depends on the distribution of demand vectors in the fixed input sequence, and is therefore difficult to analyze for such sequences arbitrarily. On the other hand, analyzing the growth of the lower bound seems possible for random input sequences, but we doubt that this would give any meaningful insight. For this reason we only give experimental evidence that this algorithm performs well, see Section 4.

3.2.1. Enumeration

As easy as the enumeration in the second phase of our RHE-algorithm looks, this should be done efficiently. So let us have a look at the problem at hand: we want to enumerate all possible $b_1, \ldots, b_d$ with sum B and individual lower and upper bounds $\ell(i), u(i) \in \{0, \ldots, B\}$ on the summands $\ell(i) \leq b_i \leq u(i), i \in \{1, \ldots, d\}$. For short, we also denote these bounds as vectors $\mathbf{l}$ and $\mathbf{u}$. In the literature on combinatorial generation algorithms such summations with upper bounds only are known as *(d)-compositions with restricted parts*, see [14] or [12]. There is a bijection to combinations of a multiset. All compositions with restricted parts can be enumerated by a *constant amortized time* (CAT) algorithm, which can be easily extended to the case with lower bounds without changing the CAT behavior. We give the modified algorithm SUMMATIONS(B, d, U) that enumerates the compositions with restricted parts in colexicographical order for convenience, and refer to [14] for its unchanged analysis. The total number of compositions with restricted parts for given $\mathbf{l}$ and $\mathbf{u}$ is the Whitney number of the second kind of the finite chain product $(\overline{u(1) - \ell(1)} + \overline{1}) \times \cdots \times (\overline{u(r) - \ell(r)} + \overline{1})$, where $\overline{x}$ denotes a chain of x elements, see again [14] for details.

Procedure SUMMATIONS (*position p, resource r, bound n*)

input : dimension d, lower bound vector $\mathbf{l}$, upper bound vector $\mathbf{u}$, sum B

output : SUMMATIONS$(B - \sum_{r=1}^{d} \ell(r), d + 1, \sum_{r=1}^{d} u(r) - \ell(r))$ evaluates all d-compositions with restricted parts for the above parameters.

```
if p = 0 then
    evaluate b
else
    for c ∈ {max(0, p − n + u(r) − ℓ(r)) ... min(u(r) − ℓ(r), p)} do
        b_r ⟵ c + ℓ(r)
        SUMMATIONS(p − c, r − 1, n − u(r) + ℓ(r))
    end
    b_r ⟵ ℓ(r)
end
```

The initial call is SUMMATIONS$(B', d + 1, U')$ for $B' = B - \sum_{r=1}^{d} \ell(r)$ and $U' = \sum_{r=1}^{d} u(r) - \ell(r)$. This algorithm has CAT behavior for $B' \leq U'/2$. For $B' > U'/2$ there is a similar algorithm that can be adapted from algorithm GEN2 in [14]. We sum up the results of this section in the following theorem.

Theorem 4. *The d-compositions with restricted parts and lower bounds needed in algorithm* RHE *can be enumerated in constant amortized time.*

3.2.2. Evaluation

For demand vectors $\mathbf{s}_i \in \mathbb{Q}_+^d, i \in \{1, \ldots, n\}$, the evaluation of a given bin vector $\mathbf{b}$, i.e., computing $\kappa(\mathbf{b}, \mathbf{S})$, can be done in the obvious way in $O(n \cdot d)$ time: Scan through the sequence starting at the last breakpoint π_ℓ (initially at $\pi_0 = 0$) updating the total demand vector $\mathbf{w}_{\pi_\ell,i}$ of the current bin until the addition of the next vector $\mathbf{s}_{i+1}$ in the sequence would make the demand vector exceed $\mathbf{b}$. Then add breakpoint $\pi_{\ell+1} = i$ and continue the scan with the next bin starting from there.

In the special case of sequential unit vector packing, the runtime can be improved to $O(n)$, since for each demand vector only one of the d resources needs to be updated and checked.

For a single evaluation of a bin vector, this algorithm is basically the best one can hope for. On the other hand, in the setting of our heuristic enumeration algorithm where many bin vectors are evaluated, the question arises whether we can speed up the evaluations if we allow for preprocessing. We describe an approach that we developed for our application, i.e., an approach for the sequential unit vector packing problem with large values of n compared to k the number of breakpoints. It is possible to extend parts of the approach to the general problem with a loss in space efficiency.

A first simple approach builds an $(n \times d \times B)$ table T_1 as sketched in Fig. 4. In this table we store in entry $T_1(p, r, \delta)$ the position of the next breakpoint in the sequence starting from position p for a bin vector $\mathbf{b}$ with capacity δ for resource r, i.e., $b_r = \delta$, and $b_i = \infty$ for $i \neq r$.

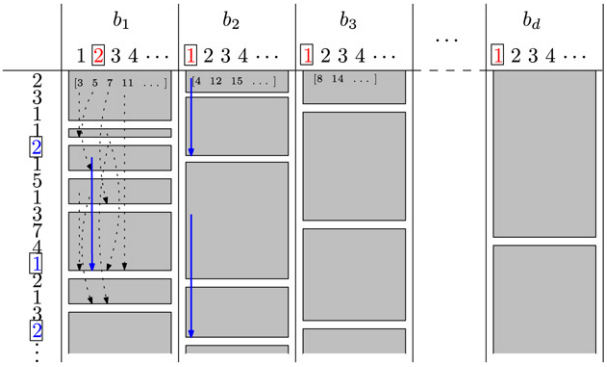


Fig. 4. Simple data structure that accelerates the evaluation of a bin vector. The rows correspond to positions in **S**. The solid arrows show the minima of Eq. (8) for the example bin vector $(2, 1, 1, \ldots, 1)$. Breakpoints are highlighted in the sequence (leftmost column). The exemplary dotted arrows indicate the end positions of the blocks, before which the relevant breakpoints are located.

To evaluate a given bin vector **b** we start at position 1 and inspect positions $(1, r, b_r)$ for $1 \le r \le d$. The next breakpoint must be at the minimum of these values. Thus we have

$$\pi_{i+1} = \min_{1 \le r \le d} T_1(\pi_i, r, b_r). \tag{8}$$

Eq. (8) directly gives an $O(kd)$ algorithm for the evaluation of a bin vector. On instances with $kd \ll n$, this is a speedup. The space complexity of this approach seems to be $\Theta(n \cdot d \cdot B)$ at first glance. But notice that between two occurrences of a resource r in the sequence the value of $T_1(\cdot, r, \cdot)$ remains the same. More precisely, if for all p' with $p_1 \le p' < p_2$ it holds that $s_{p'} \ne r$, then we have $T_1(p_1, r, \delta) = T_1(p_2, r, \delta)$ for all δ. Let us call such an interval with equal entries for a given resource r a *block*. An example can be found in Fig. 4, where the blocks are depicted as grey rectangles. The total number of blocks is bounded by $n + d = O(n)$ because at each position exactly one block ends in the setting of unit vector packing. Also the answer vectors in the blocks need not be stored explicitly: in the ith block of resource r the table entry for b_r is simply the position before the end position of block $i + b_r$ as indicated by the exemplary arrows in the figure. Therefore, in our approach we store the block structure in an array of size $O(n)$ to get a constant lookup time for a given table entry $T_1(p, r, \delta)$. More precisely, we store d arrays $\{A_1, \ldots, A_d\}$ of total size $O(n)$, such that $\{A_r(i)\}$ gives the end position of block i of resource r or equivalently the position of the ith occurrence of r in **S**. It is easy to see that the block structure can be (pre-)computed in linear time.

However, with this approach a different problem arises: after the computation of breakpoint π_{i+1}, we need to know at which positions we should access each of the arrays next. To answer this question we introduce a second table. Let T_2 be an $(n \times 2)$-table that stores in $T_2(p, 1)$ the index of the (unique) new block[1] that starts at position p and in $T_2(p, 2)$ the index of the current block of resource $(p \bmod d) + 1$ in array $A_{(p \bmod d)+1}$. In order to recompute the indices for breakpoint π_{i+1} we read the d rows $\{T_2(\pi_{i+1} - d + 1, \cdot), \ldots, T_2(\pi_{i+1}, \cdot)\}$. Each resource r occurs once in the second column of the read rows and might occur several times in the first column. Take as index for resource r the value of the last occurrence of r in the read rows, regardless of the column, i.e., the occurrence with the highest row index. This approach correctly computes all new indices in the arrays $\{A_1, \ldots, A_d\}$ in $O(d)$ time, which is also the time that a single step takes without this index computation. Obviously, table T_2 needs $O(n)$ space. Alternatively, this table T_2 can be understood as a persistent version of the list containing for every resource its next occurrence, that is updated during a scan along the sequence. In this situation, a general method for partially persistent data structures like [6] can be applied and yields the same time and space bounds. Altogether, we have shown the following theorem:

Theorem 5. *Given a sequence* **S** *and a bin vector* **b** *we can construct a data structure with* $O(n \cdot d \cdot B)$ *space and preprocessing time such that the evaluation of* $\kappa(\mathbf{b}, \mathbf{S})$ *for sequential vector packing takes* $O(\kappa(\mathbf{b}, \mathbf{S}) \cdot d)$ *time. For sequential* unit *vector packing only* $O(n)$ *space and preprocessing time is needed.*

Note that for RHE if we have already found a bin vector with k' many breakpoints we can stop all subsequent evaluations already after $k' \cdot d$ many steps.

3.3. Mixed integer linear program

Even if the ILP-formulation of Section 1.1 is a good starting point for our theoretical results, it turns out that in practice, only medium sized instances can be solved with it. One problem is the potentially quadratic number of edge flow variables

[1] Strictly speaking, the first column in table T_2 is not necessary, as it simply reproduces the sequence. Here it clarifies the presentation and the connection to the technique in [6].

Table 1
Summary information on the instances

Name	Demand unit vectors	Dimension	Window size	Note
inst1	4464	24	28	
inst1-doubled	8928	48	56	inst1 with "doubled" resources
inst2	9568	8	28	
inst3	7564	7	28	
inst4	4464	22	28	
rand-doubled	2500	26	2500	random "doubled" instance

that makes the formulation prohibitive already for small instances. To reduce the number of variables, it is helpful to have an upper bound on the length of the longest edge. One such bound is of course B, but ideally there are smaller ones. As our real-world instances are windowed instances, the window size is a trivial upper bound that helps to keep the number of variables low. A further problem is that, even if the bin vector is already determined, the MIP-solver needs to branch on the x and y variables to arrive at the final solution. This can be avoided by representing the bin vector components b_r as a sum of 0-1-variables z_i^r, such that $b_r = \sum_i z_i^r$ and $z_i^r \geq z_{i+1}^r$. If an edge e uses i units of resource r, i.e., the r-th entry of w_e is i, we include a constraint of the form $y_e \leq z_i^r$. This allows one to use the edge only if there are at least i resources available. With these additional constraints, the edge variables y and x need no longer be binary. For integral values of z_i^r, only edge-variables that do not exceed the bin vector can have a value different from zero, so that in this case every fractional path of the resulting flow is feasible with respect to the bin vector, and thus all paths have the same (optimal) number of breakpoints (otherwise a shorter path could be selected resulting in a solution with less bins). With this mixed integer linear program the number of branching nodes is drastically reduced, but the time spent at every such node is also significantly increased. Still, the overall performance of this program is much better than the original integer program, small instances (dimension 8) can now be solved to optimality within a few minutes. A bigger instance of dimension 22 and with a total bin size of up to 130 can be solved to optimality with this program within less than 3 h, for different values of the total bin size.

We observed that on this mixed integer program feasible solutions are found after a small fraction of the overall running time. Hence, we consider this approach also reasonable suited as a heuristic to come up with good solutions.

4. Experiments

Our industry partner, who has provided us with the problem setting and large real world data sets for bounded size SUVP, wishes to remain unnamed. We therefore could not give the details of the original application in this paper. But it was possible to make at least the data electronically available at www.inf.ethz.ch/personal/mnunkess/SVP/. We report on some experiments on these. We implemented the greedy algorithms, the enumeration heuristic RHE, and the integer linear program. For the latter we use CPLEX 9.0. All programs were run on a 3GHz P4 workstation with 3GB RAM, running Linux 2.4.22.

Ideally, we want to describe the quality of the heuristics by comparison to the optimal value. One attempt to compute such optimal values is to use the ILP-formulation of Section 1.1 and solve it with an ILP-solver. In our setting, this worked only for small real world instances. Medium sized instances could be solved with the ILP explained in Section 3.3. We compare solution qualities, because the running times of the different approaches are orders of magnitude apart. On many of our instances a calculation for a fixed B takes some seconds for the greedy algorithms and some hours for the mixed integer linear program. We let the enumeration heuristic run for 10 min which seems like a realistic maximum time that an "on-line" user would be willing to wait for a result. We then fix the number of repetitions of the guessing phase of RHE to be as many as it takes to let $\|\mathbf{t}\|_1$, the sum of the guessed lower bounds, exceed some fraction of B. This fraction is initially chosen as 99%, and adaptively decreased after each run as long as the targeted time of 10 min is not exceeded. The subsequence length is initially fixed with respect to the estimated number of breakpoints that we get by running both greedy approaches. We set it to one half times the average distance between two breakpoints and increase it adaptively if the lower bound does not grow any more after a fixed number of repetitions in the initialization phase. By the adaptive choice of both the subsequence length and the fraction the algorithm is robust with respect to changing values of B, d and the time that it is run.

In Fig. 5 we show the relative performances on our biggest real world instance (Inst1), cf. Table 1, which summarizes information on our instances. The different data points correspond to the algorithms Greedy-Grow, Greedy-Shrink, RHE and the linear relaxations of the two different ILP formulations. The values represent the ratio of the solution found by the algorithm to the optimal integral solution that we calculated using the MIP formulation. The figure shows that for small values of B Greedy-Grow produces results that are close to optimal, whereas for bigger values the quality gets worse. An explanation for this behavior is that Greedy-Grow builds a solution iteratively. As the results of Section 3 show, the greedy algorithms can be forced to take decisions based only on the tie-breaking rule. On this instance tie-breaking is used at several values of B, which leads to an accumulation of errors, in addition to the inherent heuristic nature of the method. Note that by definition Greedy-Shrink is optimal for $B = \|\sum_{i=1}^{n} \mathbf{s}_i\|_1$, which is 196 on this instance. In order to have a meaningful scale we let the x-axis stop before that value.

In Fig. 6(a) we present the quality of the solutions delivered by RHE relative to the optimal solution on four further instances. Note that for different instances, different values of B make sense. Instance Inst1-doubled is obtained from Inst1, by the doubling transformation used in the proof of Observation 1. In Fig. 6(b) we compare the best of the two greedy results

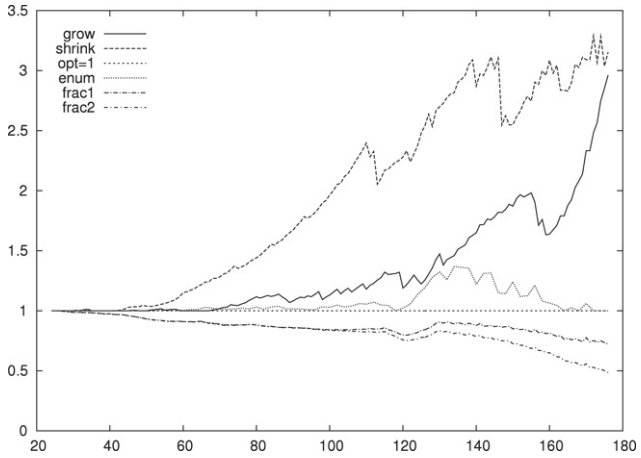
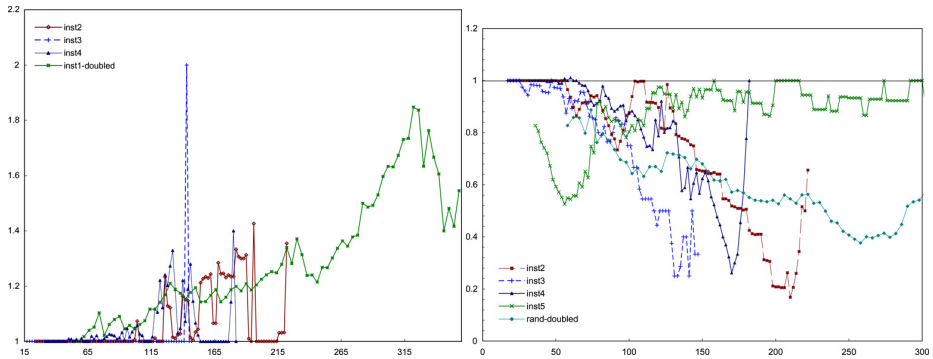


Fig. 5. Plot of ratio of solution value of the different approaches to optimal solution value versus total bin size B.



(a) Ratio of RHE to optimal solution on 5 instances. (b) Ratio of RHE to best of greedy algorithms on 5 instances.

Fig. 6. Results on the instances of Table 1

to the result of RHE. (Note that even if we use the greedy algorithms to determine the parameter settings of RHE, these results are not visible to RHE.) Instance rand-doubled is an instance where first the demand unit vectors are drawn uniformly at random, and then a doubling transformation is performed to make it potentially more complicated. It could not be solved to optimality by the MIP approach, and does therefore not occur in Fig. 6(a). Compared to the other instances the greedy algorithms do not perform too badly. One reason for this is the uniform distribution for the resources. It follows that the tie-breaking rules make the right choices "on average". On the other hand, on the doubled real-world instance Inst1-doubled RHE gives superior results and in particular for higher values of B the greedy algorithms perform comparatively poorly.

5. Conclusion

In this paper, we have introduced the sequential vector packing problem, presented $\mathcal{NP}$-hardness proofs for different variations, two approximation algorithms, several heuristics, and an experimental evaluation. For our industry application, both the presented enumeration heuristic and the greedy algorithms helped to drastically improve the existing solution. The MIP approach is interesting, because it can compute the optimal solution for moderate size instances. The approximation algorithms give insights into the combinatorial structure of the problem, as well as providing guaranteed quality solutions quickly. From our point of view, the most interesting challenges are to find an approximation algorithm for bounded size SVP or an inapproximability result.

References

[1] E. Coffman Jr, M.R. Garey, D.S. Johnson, Approximation algorithms for bin packing: An updated survey, in: Algorithm Design for Computer System Design, Springer, 1984, pp. 49–106.
[2] E. Coffman Jr, M.R. Garey, D.S. Johnson, Approximation algorithms for bin packing: A survey, in: Approximation Algorithms, PWS Publishing Company, 1997, pp. 46–93.
[3] D. Coppersmith, P. Raghavan, Multidimensional on-line bin packing: Algorithms and worst-case analysis, Operations Research Letters 4 (1989) 48–57.
[4] E. Dantsin, A. Goerdt, E.A. Hirsch, R. Kannan, J.M. Kleinberg, C.H. Papadimitriou, P. Raghavan, U. Schöning, A deterministic (2-2/(k+1))n algorithm for k-sat based on local search, Theoretical Computer Science 289 (1) (2002) 69–83.
[5] R.G. Downey, M.R. Fellows, Parametrized complexity, in: Monographs in Computer Science, Springer, 1999.

[6] J.R. Driscoll, N. Sarnak, D.D. Sleator, R.E. Tarjan, Making data structures persistent, in: STOC '86: Proceedings of the 18th Annual ACM Symposium on Theory of Computing, ACM Press, New York, NY, USA, 1986, pp. 109–121.
[7] U. Feige, D. Peleg, G. Kortsarz, The dense k-subgraph problem, Algorithmica 29 (3) (2001) 410–421.
[8] G. Galambos, G.J. Woeginger, On-line bin packing—a restricted survey, Mathematical Methods of Operations Research 42 (1) (1995) 25–45.
[9] M.R. Garey, D.S. Johnson, Computers and Intractability, Freeman, 1979.
[10] B. Kalyanasundaram, K. Pruhs, Speed is as powerful as clairvoyance, Journal of the ACM 47 (2000) 617–643.
[11] R. Niedermeier, Invitation to Fixed-Parameter Algorithms, Oxford University Press, 2006.
[12] A. Nijenhuis, H. Wilf, Combinatorial Algorithms, 2nd ed., Academic Press, 1978.
[13] C. Phillips, C. Stein, E. Torng, J. Wein, Optimal time-critical scheduling via resource augmentation, in: Proceedings of the 29th Annual ACM Symposium on Theory of Computing, STOC, 1997, pp 140–149.
[14] F. Ruskey, Combinatorial generation, 2005.
[15] U. Schöning, A probabilistic algorithm for k-SAT based on limited local search and restart, Algorithmica 32 (2002) 615–623.
[16] T.S. Wee, M.J. Magazine, Assembly line balancing as generalized bin-packing, Operations Research Letters 1 (1982) 56–58.
[17] J. Yang, J.Y.-T. Leung, The ordered open-end bin-packing problem, Operations Research 51 (5) (2003) 759–770.