Sequential Vector Packing*

Mark Cieliebak[2], Alexander Hall[1], Riko Jacob[1], and Marc Nunkesser[1]

[1] Department of Computer Science, ETH Zurich, Switzerland
{alex.hall,riko.jacob,marc.nunkesser}@inf.ethz.ch
[2] sd&m Schweiz AG, 8050 Zurich, Switzerland
mark.cieliebak@sdm.com

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 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 used bins 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 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.

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 $i \in \{\mathrm{A} \ldots \mathrm{Z}\}$ 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 $(b_\mathrm{A}, \ldots, b_\mathrm{Z})$) 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

* 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.

B. Chen, M. Paterson, and G. Zhang (Eds.): ESCAPE 2007, LNCS 4614, pp. 12–23, 2007.

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:

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*

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

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* j*th 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* i*th 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 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 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 [2, 8] 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 [13, 12] that remotely resemble our setting. Galambos and Woeginger [6] 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. [9, 10].

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 randomized 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.Due to space limitations, we omit some of the proofs in this extended abstract. These proofs can be found in the technical report [1], in which we also experimentally evaluate the presented algorithms with promising results on real world data, in particular for the randomized heuristic.

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 $\mathbf{b}$. 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.

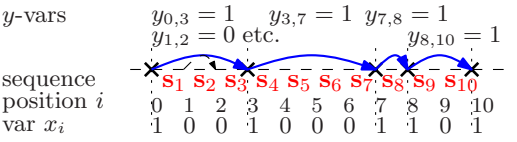

Fig. 1. An exemplary instance, its potential ILP solution, and its flow representation. The solution needs 4 bins.

$$\begin{aligned}
&\min\ k \text{ (or } B) \\
&\text{s.t. } x_0 = 1 && (1)\\
&\sum_{u=0}^{i-1} y_{u,i} = x_i \text{ for } i \in \{1,\dots,n\} && (2)\\
&\sum_{v=i+1}^{n} y_{i,v} = x_i \text{ for } i \in \{0,\dots,n-1\} && (3)\\
&\sum_{\substack{u,v:\\ u<i\le v}} \mathbf{w}_{u,v}\cdot y_{u,v} \le \mathbf{b} \text{ for } i \in \{1,\dots,n\} && (4)\\
&\sum_{j=1}^{d} b_j = B, \qquad \sum_{i=1}^{n} x_i = k && (5)\\
&B \in \mathbb{Q}_+, (k \in \mathbb{N}), \mathbf{b} \in \mathbb{Q}_+^d, x_i, y_{u,v} \in \{0,1\} \text{ for } x_i \in X, y_{u,v} \in Y
\end{aligned}$$

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 $u \in \{0,\dots,n-1\}$ to position $v \in \{1,\dots,n\}$, with $u < v$, see Figure 1.

The Constraints (2),(3) 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 (4) 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 (4) 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 (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.

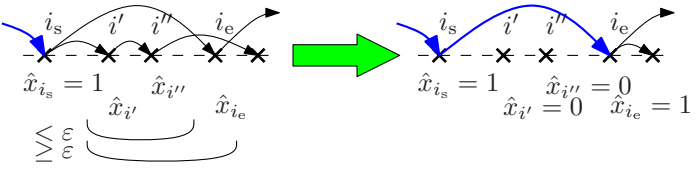

Fig. 2. An example of the rerouting of flow in Lines 4 (a)-(c) of the algorithm

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 Figure 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.*

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, see Section 3.

1.3 A 2-Approximation for Bounded Number Sequential Vector Packing

We prove some properties of the LP relaxation and then describe how they can be applied to obtain the rounding scheme yielding the desired bicriteria ratio.

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 of $e_1, \hat{e}_1$). 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.

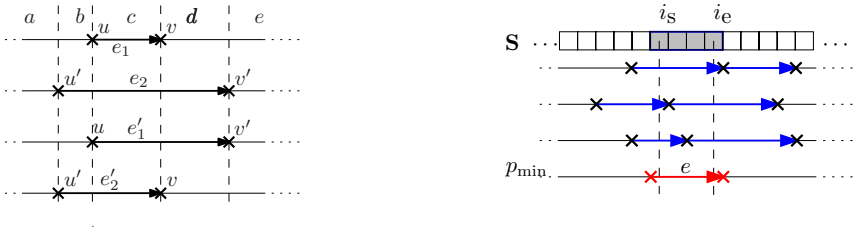

Fig. 3. Left: Replacement of λ units of flow on e_1 and e_2 by λ units of flow on e'_1 and e'_2 in Lemma 1. Right: Extracting the integral solution. Edge e together with other potential edges in $Y^\star$ in Theorem 2.

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 Figure 3 (left). The remaining flow values in Y' are taken directly from Y. It is easy to see that the flow conservation constraints (2),(3) 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 (4) 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 (4) 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 Figure 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:

Region b (d). The only variables in (4) 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 (4) still holds in region b. Region d is analogous to b.

Region c. Here the only variables which change in (4) 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 (4) 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 (4) 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. 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 $O(n^2)$. □

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 convex 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 algorithm presented 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 (4). Figure 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 $E_{i_\mathrm{s}} = \{(u, v) | 0 \leq u < i_\mathrm{s} \leq v \leq n\}$ (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 $E_{i_\mathrm{s}}, (E_{i_\mathrm{e}})$. Furthermore, let $E' = E_{i_\mathrm{s}} \cup E_{i_\mathrm{e}}$. Summing up the two Constraints (4) for i_s and i_e gives $2\mathbf{b}^\star \geq \sum_{(u,v) \in E_{i_\mathrm{s}}} y_{u,v}^\star \cdot \mathbf{w}_{u,v} + \sum_{(u,v) \in E_{i_\mathrm{e}}} y_{u,v}^\star \cdot \mathbf{w}_{u,v} =: A$ and

thus $2\mathbf{b}^\star \geq A \geq \sum_{i_{\min}<i\leq i_{\max}} \sum_{\substack{(u,v)\in E':\\ u<i\leq v}} y^\star_{u,v} \cdot \mathbf{s}_i \geq \sum_{i_s<i\leq i_e} \sum_{\substack{(u,v)\in E':\\ u<i\leq v}} y^\star_{u,v} \cdot \mathbf{s}_i = \sum_{i_s<i\leq i_e} \mathbf{s}_i = \mathbf{w}_{i_s,i_e}$. The second inequality is in general an inequality because the sets E_{i_s} and E_{i_e} need not be disjoint. For the third inequality 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. 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. 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-Completeness

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, always after w positions. 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 $j \in \{1, \ldots, w\}$, denoted as $s^i_j \in \{1, \ldots, d\}$. 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 Clique [7] or more generally from k-densest subgraph [5]. 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, i.e., $V = \{1, .., d\}$. 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 chose 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, or decrease the number of resources to two. 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 such 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).

We introduce two different stopping sequences F and G which we use alternatingly. This ensures 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 used numbers are polynomial in the size of the original graph. □

For a discussion of polynomially solvable cases and issues related to fixed parameter tractibility, see [1].

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.

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). 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 the light of the following observations (for proofs see [1]) 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.*

It follows that Greedy-Grow with an unspecified tie-breaking rule can be led to produce arbitrarily bad solutions. Also Greedy-Shrink can produce bad solutions depending on the tie-breaking scheme.

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.*

For the experiments in [1] 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.

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 [11] 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):

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.

It is straight-forward 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. We give experimental evidence that the algorithm performs well and present an efficient algorithm for the enumeration in Phase 2, see [1].

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. With a preprocessing phase and some algorithmic engineering we can show the following theorem (see [1] for a complete discussion).

Theorem 4. *Given a sequence* $\mathbf{S}$ *and a bin vector* $\mathbf{b}$ *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.*

4 Conclusion

In this paper, we have introduced the sequential vector packing problem, presented $\mathcal{NP}$-hardness proofs for different variants, approximation algorithms, and several heuristics. The most interesting open challenges are probably to find an approximation algorithm for bounded size SVP and inapproximability results.

References

[1] Cieliebak, M., Hall, A., Jacob, R., Nunkesser, M.: Sequential vector packing. DELIS TR 0335, ETH Zurich (2006)
[2] Coffman Jr., E., Garey, M.R., Johnson, D.S.: Algorithm Design for Computer System Design. In: Approximation Algorithms for Bin Packing: An updated Survey, pp. 49–106. Springer, Heidelberg (1984)
[3] Coppersmith, D., Raghavan, P.: Multidimensional on-line bin packing: Algorithms and worst-case analysis. Operations Research Letters 4, 48–57 (1989)
[4] Dantsin, E., Goerdt, A., Hirsch, E.A., Kannan, R., Kleinberg, J.M., Papadimitriou, C.H., Raghavan, P., Schöning, U.: A deterministic (2-2/(k+1))n algorithm for k-sat based on local search. Theoretical Computer Science 289(1), 69–83 (2002)
[5] Feige, U., Peleg, D., Kortsarz, G.: The dense k-subgraph problem. Algorithmica 29(3), 410–421 (2001)
[6] Galambos, G., Woeginger, G.J.: On-line bin packing—a restricted survey. Mathematical Methods of Operations Research 42(1), 25–45 (1995)
[7] Garey, M.R., Johnson, D.S.: Computers and Intractability. Freeman, San Francisco (1979)
[8] Approximation Algorithms. In: Hochbaum, D.S. (ed.) Approximation Algorithms For Bin Packing: A Survey, pp. 46–93. PWS Publishing Company (1997)
[9] Kalyanasundaram, B., Pruhs, K.: Speed is as powerful as clairvoyance. Journal of the ACM 47, 617–643 (2000)
[10] Phillips, C., Stein, C., Torng, E., Wein, J.: Optimal time-critical scheduling via resource augmentation. In: STOC. Proceedings of the 29th Annual ACM Symposium on Theory of Computing, pp. 140–149. ACM Press, New York (1997)
[11] Schöning, U.: A probabilistic algorithm for k-SAT based on limited local search and restart. Algorithmica 32, 615–623 (2002)
[12] Wee, T.S., Magazine, M.J.: Assembly line balancing as generalized bin-packing. Operations Research Letters 1, 56–58 (1982)
[13] Yang, J., Leung, J.Y.-T.: The ordered open-end bin-packing problem. Operations Research 51(5), 759–770 (2003)