A Faster FPTAS for the Unbounded Knapsack Problem

Klaus Jansen    Stefan E. J. Kraft
Department of Computer Science, University of Kiel
{kj, stkr}@informatik.uni-kiel.de

Abstract

The Unbounded Knapsack Problem (UKP) is a well-known variant of the famous 0-1 Knapsack Problem (0-1 KP). In contrast to 0-1 KP, an arbitrary number of copies of every item can be taken in UKP. Since UKP is NP-hard, fully polynomial time approximation schemes (FPTAS) are of great interest. Such algorithms find a solution arbitrarily close to the optimum \( \text{OPT}(I) \), i.e. of value at least \((1-\varepsilon)\text{OPT}(I)\) for \(\varepsilon > 0\), and have a running time polynomial in the input length and \(\frac{1}{\varepsilon}\). For over thirty years, the best FPTAS was due to Lawler with running time in \(O(n + \frac{1}{\varepsilon^3})\) and space complexity in \(O(n + \frac{1}{\varepsilon^2})\), where \(n\) is the number of knapsack items. We present an improved FPTAS with running time \(O(n + \frac{1}{\varepsilon^2} \log^3 \frac{1}{\varepsilon})\) and space bound \(O(n + \frac{1}{\varepsilon} \log^2 \frac{1}{\varepsilon})\).

This directly improves the running time of the fastest known approximation schemes for Bin Packing and Strip Packing, which have to approximately solve UKP instances as subproblems.

1 Introduction

An instance \(I\) of the Knapsack Problem (KP) consists of a list of \(n\) items \(a_1, \ldots, a_n\), \(n \in \mathbb{N}\), where every item has a profit \(p_j \in (0, 1]\) and a size \(s_j \in (0, 1]\). Moreover, we have the knapsack size \(c = 1\). In the 0-1 Knapsack Problem (0-1 KP), a subset \(V \subset \{a_1, \ldots, a_n\}\) has to be chosen such that the total profit of \(V\) is maximized and the total size of the items in \(V\) is at most \(c\). Mathematically, the problem is defined by \(\max\{\sum_{j=1}^{n} p_j x_j | \sum_{j=1}^{n} s_j x_j \leq c; x_j \in \{0, 1\} \forall j\}\). In this paper, we focus on the unbounded variant (UKP) where an arbitrary number of copies of every item is allowed, i.e. \(\max\{\sum_{j=1}^{n} p_j x_j | \sum_{j=1}^{n} s_j x_j \leq c; x_j \in \mathbb{N} \forall j\}\).

1.1 Known Results

The 0-1 knapsack problem and its variants are well-known NP-hard problems [5]. They can be optimally solved in pseudo-polynomial time by dynamic programming [1, 18]. Furthermore, fully polynomial time approximation schemes (FPTAS) are known for different variants of KP. An FPTAS is a family of algorithms \((A_{\varepsilon})_{\varepsilon > 0}\), where for every \(\varepsilon > 0\) the algorithm \(A_{\varepsilon}\) finds for a given instance \(I\) a solution of profit \(A_{\varepsilon}(I) \geq (1-\varepsilon)\text{OPT}(I)\). The value \(\text{OPT}(I)\) denotes the optimal value for \(I\). FPTAS have a running time polynomial in \(\frac{1}{\varepsilon}\) and the input length.

The first FPTAS for 0-1 KP was presented by Ibarra and Kim [8]. The running time was improved by Lawler in his seminal paper [20]. The currently fastest known algorithm is by Kellerer

\[\text{OPT}(I) = \max\{\sum_{j=1}^{n} p_j x_j | \sum_{j=1}^{n} s_j x_j \leq c; x_j \in \{0, 1\} \forall j\}\]
and Pferschy [16, 17, 18] pp. 166–183] with a space complexity in $O(n + \frac{1}{\varepsilon^2})$ and a running time in $O(n \min\{\log n, \log \frac{1}{\varepsilon}\} + \frac{1}{\varepsilon^2} \log(\frac{1}{\varepsilon}) \cdot \min\{n, \frac{1}{\varepsilon} \log(\frac{1}{\varepsilon})\}$). Assuming that $n \in \Omega(\frac{1}{\varepsilon} \log(\frac{1}{\varepsilon}))$, this is in $O(n \log(\frac{1}{\varepsilon}) + \frac{1}{\varepsilon^2} \log^2(\frac{1}{\varepsilon})$)

For UKP, Ibarra and Kim [8] presented the first FPTAS by extending their 0-1 KP algorithm. Their UKP algorithm has a running time in $O(n + \frac{1}{\varepsilon} \log \frac{1}{\varepsilon})$ and a space complexity in $O(n + \frac{1}{\varepsilon} \log \frac{1}{\varepsilon})$. Kellerer et al. [18] pp. 232–234] have moreover described an FPTAS with running time in $O(n \log(n) + \frac{1}{\varepsilon^2} (n + \log \frac{1}{\varepsilon}))$ and a space bound in $O(n + \frac{1}{\varepsilon})$. In 1979, Lawler [20] presented his FPTAS with running time in $O(n + \frac{1}{\varepsilon})$ and space complexity in $O(n + \frac{1}{\varepsilon})$. For $n \in \Omega(\frac{1}{\varepsilon})$, this is still the best FPTAS.

The study of KP is not only interesting in itself, it is moreover motivated by column generation for optimization problems like the famous Bin Packing Problem and Strip Packing Problem. In the former problem, a set $J$ of $n$ items of size in $(0, 1]$ has to be packed in as few unit-sized bins as possible. In the latter problem, a set $J$ of $n$ rectangles of width $(0, 1]$ and height $(0, 1]$ has to be packed in a strip of unit width such that the height is minimized. Many algorithms for optimization problems like Bin Packing have to solve linear programs (LPs), but enumerating all columns of the linear programs would take too much time. One way to avoid this is the consideration of the dual of the LP and to (approximately or exactly) solve a separation problem, e.g. KP, to find violated inequalities of the dual. These inequalities correspond to columns in the primal LP: the columns needed for solving the LP are therefore generated and added dynamically. Examples can be found in [6, 14].

Since Bin Packing and Strip Packing are NP-complete [5], several approximation algorithms have been found for them. However, the best absolute approximation ratio for both problems is $\frac{2}{3}$ for all efficient (i.e. polynomial-time) algorithms, unless $P = NP$ [5]. So-called asymptotic fully polynomial-time approximation schemes (AFPTAS) $(A_\varepsilon)_{\varepsilon > 0}$ are therefore especially interesting. They find for every $\varepsilon > 0$ and instance $J$ a solution of value of at most $(1 + \varepsilon)OPT(J) + f(\frac{1}{\varepsilon})$, and have a running time polynomial in $n$ and $\frac{1}{\varepsilon}$. Roughly speaking, the asymptotic approximation ratio for an AFPTAS can be seen as the approximation ratio achieved for large instances.

For Bin Packing, the first AFPTAS was presented by Karmarkar and Karp [14] with $f(\frac{1}{\varepsilon}) = O(\frac{1}{\varepsilon^2})$. In 1991, Plotkin et al. [22] described an improved algorithm with a smaller additive term $f(\frac{1}{\varepsilon}) = O(\frac{1}{\varepsilon} \log(\frac{1}{\varepsilon}))$ and running time in $O(\frac{1}{\varepsilon^2} \log^6(\frac{1}{\varepsilon}) + \log(\frac{1}{\varepsilon}) n)$. The AFPTAS by Shachnai and Yehezkel [23] has the same additive term and a running time in $O(\frac{1}{\varepsilon^2} \log^3(\frac{1}{\varepsilon}) + \log(\frac{1}{\varepsilon}) n)$ for general instances. Currently, the AFPTAS in [10] has the smallest additive term $f(\frac{1}{\varepsilon}) = O(\log^2(\frac{1}{\varepsilon}))$ and the fastest running time in $O(\frac{1}{\varepsilon^2} \log^2(\frac{1}{\varepsilon}) + \log(\frac{1}{\varepsilon}) n)$.

The first AFPTAS for Strip Packing was presented by Kenyon and Rémila [19] with additive term $f(\frac{1}{\varepsilon}) = O(\frac{1}{\varepsilon^2})$. Bougeret et al. [2] and Sviridenko [24] independently improved the additive term to $f(\frac{1}{\varepsilon}) = O(\frac{1}{\varepsilon} \log(\frac{1}{\varepsilon}))$. The running time is in $O(\frac{1}{\varepsilon^2} \log(\frac{1}{\varepsilon}) + n \log n)$ [2], which is the currently fastest known AFPTAS.

Both algorithms in [2, 10] solve UKP instances for column generation. A faster FPTAS for UKP therefore directly yields faster AFPTAS for Bin Packing and Strip Packing.

1.2 Our Result

We have derived an improved FPTAS for UKP that is faster and needs less space than Lawler’s algorithm.
Theorem 1. There is an FPTAS for UKP with running time in $O(n + \frac{1}{\varepsilon} \log^{3}(\frac{1}{\varepsilon}))$ and required storage space in $O(n + \frac{1}{\varepsilon} \log^{2}(\frac{1}{\varepsilon}))$.

Not only the improved running time, but also the improved space complexity is interesting because “for higher values of $\frac{1}{\varepsilon}$ the space requirement is usually considered to be a more serious bottleneck for practical applications than the running time” [18] p. 168]. Nevertheless, the improved time complexity has direct practical consequences. Let $KP(d, \varepsilon)$ be the running time to find a $(1 - \varepsilon)$ approximate solution to a UKP instance with $d$ items. The Bin Packing algorithm in [10] has the running time $O(KP(d, \frac{\varepsilon}{6}) \cdot \frac{1}{\varepsilon^3} \log \frac{1}{\varepsilon} + \log(\frac{1}{\varepsilon}))$ if we assume that $KP(d, \frac{\varepsilon}{6}) \in \Omega(\frac{1}{\varepsilon^3})$ (where $\varepsilon \in \Theta(\varepsilon)$ and $d \in O(\frac{1}{\varepsilon} \log \frac{1}{\varepsilon})$). By using the new FPTAS for UKP, we get the following result:

Corollary 2. There is an AFPTAS $(A_\varepsilon)_{\varepsilon>0}$ for Bin Packing that finds for $\varepsilon \in (0, \frac{1}{2}]$ a packing of $J$ in $A_\varepsilon(J) \leq (1 + \varepsilon)OPT(J) + O(\log^{2}(\frac{1}{\varepsilon}))$ bins. Its running time is in

$$O\left(\frac{1}{\varepsilon^3} \log \frac{1}{\varepsilon} + \log\left(\frac{1}{\varepsilon}\right) n\right).$$

Similarly, the Strip Packing algorithm in [2] (see also [9]) has a running time in $O(d(\frac{1}{\varepsilon^2} + \ln d) \max\{KP(d, \varepsilon), d \ln \ln(\varepsilon)\} + n \log n)$ where again $d \in O(\frac{1}{\varepsilon} \log \frac{1}{\varepsilon})$ and $\varepsilon \in \Theta(\varepsilon)$. The new FPTAS yields the following improved AFPTAS:

Corollary 3. There is an AFPTAS $(A_\varepsilon)_{\varepsilon>0}$ for Strip Packing that finds a packing of $J$ of total height at most $A_\varepsilon(J) \leq (1 + \varepsilon)OPT(J) + O(\frac{1}{\varepsilon} \log(\frac{1}{\varepsilon}))$. Its running time is in

$$O\left(\frac{1}{\varepsilon^3} \log \frac{1}{\varepsilon} + \log\left(\frac{1}{\varepsilon}\right) n\right).$$

For readers acquainted with column generation or linear programs, it should be noted that the LP solved has the form $\min\{c^T x \mid Ax \geq b, x \geq 0\}$. It is indeed a fractional covering problem where the columns of $A$ represent configurations: a configuration assigns item slots to one bin (for Bin Packing) or to one shelf of the strip (for Strip Packing) such that the items fit into the bin or the strip. The primal LP is then approximately solved with a method by Grigoriadis et al. [7] (see also [9]). The columns (i.e. configurations) are generated by solving so-called block problems, i.e. UKP instances. Because of the unboundedness, some configurations may indeed assign more item slots of a certain size to one bin or the strip than there are items in the considered Bin or Strip Packing instance. This does not represent a problem because the supernumerary item slots are simply left empty in the final solution. For comparison, Plotkin et al. [22] solve the LP with a decomposition method where the block problem has additional constraints on the knapsack variables: it is a Bounded Knapsack Problem where a limited number $d_j \in \mathbb{N}$ of copies for every item $a_j$ may be taken.

1.3 Techniques

Most algorithms for UKP in [8, 18, 20] rely on 0-1 KP algorithms. The 0-1 KP algorithms determine a first lower bound $P_0$ for $OPT(I)$. Based on a threshold $T$ depending on $P_0$, the items are partitioned into large(-profit) items with $p_j \geq T$ and small(-profit) items with $p_j < T$. A reduced subset of large items is taken, whose profits are then scaled and the well-known dynamic programming by
profits applied on them. All combinations of large items and small items (which are greedily added) are checked and the best returned. For UKP, copies of the items in the reduced large item set are taken to transform the UKP instance into a 0-1 KP instance.

Our algorithm also first reduces the number of large items. However, we further preprocess the large items by taking advantage of the unboundedness: large items of similar profit \([2^k T, 2^{k+1} T]\) are iteratively combined (“glued”) together to larger items. Apart from two special cases that can be easily solved, we prove for this new set \(\tilde{I}\) a structure property: there are approximate solutions where at most one large item from every interval \([2^k T, 2^{k+1} T]\) is used, i.e. only \(O(\log \frac{1}{\epsilon})\) items. As a next step, a large item \(a_{\text{eff} - c}\) that consists of several copies of the most efficient small item is introduced. We prove that there are now approximate solutions for the large items \(\tilde{I} \cup \{a_{\text{eff} - c}\}\) that consist of \(O(\log \frac{1}{\epsilon})\) large items and that additionally have at least one item of profit at least \(\frac{1}{3} P_0\). Instead of standard dynamic programming, we use approximate dynamic programming: the profits in \([\frac{1}{4} P_0, 2 P_0]\) are divided into intervals of equal length. During the execution of the dynamic program, we eliminate dominated solutions and store for each interval at most one solution of smallest size. The combination of approximate dynamic programming with the structure properties yields the considerable improvement in running time and space complexity. The algorithm then returns the best combination of solutions for large items (packed by the dynamic program) and small items (added greedily).

2 Preliminaries

We introduce some useful notation. The profit of an item \(a\) is denoted by \(p(a)\) and its size by \(s(a)\). If \(a = a_j\), we also write \(p(a_j) = p_j\) and \(s(a_j) = s_j\). Let \(V = \{x_a : a \mid a \in I\}\) be a multiset of items, i.e. a subset of items in \(I\) with their multiplicities. We naturally define the total profit \(p(V) := \sum_{x_a > 0} p(a) x_a\) and the total size \(s(V) := \sum_{x_a > 0} s(a) x_a\).

Let \(v \leq c = 1\) be a part of the knapsack. The corresponding optimum profit is denoted by \(OPT(I,v) = \max\{\sum_{a \in I} p(a)x_a \mid \sum_{a \in I} s(a)x_a \leq v; x_a \in \mathbb{N}\}\). Obviously, \(OPT(I) = OPT(I, c)\) holds.

Finally, we assume throughout the paper that basic arithmetic operations as well as computing the logarithm can be performed in \(O(1)\).

2.1 A First Approximation

We present a simple approximation algorithm for \(OPT(I)\). Take the most efficient item \(a_{\text{meff}} := \arg \max_{a \in I} \frac{p(a)}{s(a)}\). Fill the knapsack with as many copies of \(a_{\text{meff}}\) as possible, i.e. take \(\lfloor \frac{c}{s(a_{\text{meff}})} \rfloor\) copies of \(a_{\text{meff}}\). Then the following holds (proof taken from [18, p. 232, 20]):

**Theorem 4.** We have \(P_0 := p(a_{\text{meff}}) \cdot \lfloor \frac{c}{s(a_{\text{meff}})} \rfloor \geq \frac{1}{2} OPT(I)\). The value \(P_0\) can be found in time \(O(n)\) and space \(O(1)\).

**Proof.** Assume first that \(a_{\text{meff}}\) can completely greedily fill the knapsack. Then \(p(a_{\text{meff}}) \cdot \lfloor \frac{c}{s(a_{\text{meff}})} \rfloor = OPT(I)\). Otherwise, one additional item \(a_{\text{meff}}\) exceeds the capacity \(c\). Then \(p(a_{\text{meff}}) \cdot \lfloor \frac{c}{s(a_{\text{meff}})} \rfloor + p(a_{\text{meff}}) \geq OPT(I)\). If \(p(a_{\text{meff}}) \leq \frac{1}{2} OPT(I)\), then \(p(a_{\text{meff}}) \cdot \lfloor \frac{c}{s(a_{\text{meff}})} \rfloor \geq OPT(I) - p(a_{\text{meff}}) \geq \frac{1}{2} OPT(I)\).
\( \frac{1}{2} \OPT(I) \), and the theorem follows. Otherwise \( p(a_{\text{meff}}) \cdot \left\lfloor \frac{\varepsilon}{s(a_{\text{meff}})} \right\rfloor \geq p(a_{\text{meff}}) \geq \frac{1}{2} \OPT(I) \), which also proves the theorem.

To determine \( P_0 \), we only have to check all items (which can be done in \( O(n) \)) and to save the most efficient item (which only needs \( O(1) \)).

From now on, we assume without loss of generality that \( \varepsilon \leq \frac{1}{4} \) and \( \varepsilon = \frac{1}{2^\kappa} \) for \( \kappa \in \mathbb{N} \). Otherwise, we replace \( \varepsilon \) by the corresponding \( \frac{1}{2^\kappa} \) such that \( \frac{1}{2^\kappa} \leq \varepsilon < \frac{1}{2^{\kappa-1}} \). Note that \( \log_2(\frac{2}{\varepsilon}) = \kappa \) holds.

Similar to Lawler \[20\], we introduce the threshold \( T \) and a modified constant \( K \):

\[
T := \frac{1}{2} \varepsilon P_0 = \frac{1}{2} \frac{2^\kappa - 1}{2^\kappa - 1} P_0
\]

and

\[
K := \frac{1}{4} \log_2(\frac{2}{\varepsilon}) + 1 = \frac{1}{4} \frac{1}{\kappa + 1} \frac{1}{2^\kappa - 1} T = \frac{1}{8} \frac{1}{\kappa + 1} \left( \frac{1}{2^\kappa - 1} \right)^2 P_0 .
\]

We will see later that these values are indeed the right choice for the algorithm.

### 3 Reducing the Items

We first partition the items into large(-profit) and small(-profit) items, and only keep the most efficient small item:

\[
I_L := \{ a \in I \mid p(a) \geq T \} , \quad I_S := I \setminus I_L , \quad \text{and} \quad a_{\text{eff}} := \arg \max \left\{ \frac{p(a)}{s(a)} \mid p(a) < T \right\} .
\]

**Theorem 5.** \( I_L \) and the item \( a_{\text{eff}} \) can be found in time \( O(n) \) and space \( O(n) \). This is also the space needed to save \( I_L \).

**Proof.** Obvious.

Similar to Lawler, we now reduce the item set \( I_L \). First, we partition the interval of large item profits \([T, 2P_0]\) into

\[
L^{(k)} := [2^k T, 2^{k+1} T) \quad \text{for} \quad k = 0, \ldots, \kappa + 1 .
\]

Note that

\[
L^{(\kappa)} = [2^\kappa T, 2^{\kappa+1} T) = \left[ \frac{2^\kappa 1}{2^{\kappa-1}} P_0, 2^{\kappa+1} \frac{1}{2^{\kappa-1}} P_0 \right) = [P_0, 2P_0) .
\]

For convenience, we directly set \( L^{(\kappa+1)} := \{2P_0\} \).

We further split the \( L^{(k)} \) into disjoint sub-intervals, each of length \( 2^k K \):

\[
L_{\gamma}^{(k)} := [2^k T + \gamma \cdot 2^k K, 2^k T + (\gamma + 1) 2^k K) \quad \text{for} \quad \gamma = 0, \ldots, 2^{\kappa+1}(\kappa + 1) - 1 .
\]

Note that indeed \( L^{(k)} = \bigcup_{\gamma} L_{\gamma}^{(k)} \) holds because

\[
2^k T + (\gamma + 1) 2^k K |_{\gamma=2^{\kappa+1}(\kappa+1)-1} = 2^k T + 2^{\kappa+1}(\kappa + 1) 2^k K
\]

\[
= 2^k T + 2^{\kappa+1}(\kappa + 1) 2^k \frac{1}{2^\kappa-1} \frac{1}{2^\kappa - 1} T
\]

\[
= 2^k T + 2^k T = 2^{k+1} T .
\]
Similar to above, we set $L_0^{(\kappa+1)} := \{2P_0\}$.

The idea is to keep only the smallest item $a$ for every profit interval $L^{(k)}_\gamma$. We will see that these items are sufficient to determine an approximate solution.

**Definition 6.** For an item $a$ with $p(a) \geq T$, let $k(a) \in \mathbb{N}$ be the interval such that $p(a) \in L^{(k(a))}_\gamma$ and $\gamma(a) \in \mathbb{N}$ be the sub-interval such that $p(a) \in L^{(k(a))}_{\gamma(a)}$. Let $a^{(k)}_\gamma$ be the smallest item for profit interval $L^{(k)}_\gamma$, i.e.

$$a^{(k)}_\gamma := \arg \min \left\{ s(a) \mid a \in I_L \text{ and } p(a) \in L^{(k)}_\gamma \right\} \text{ for all } k \text{ and } \gamma.$$ 

Algorithm 1 shows the algorithm to determine the $a^{(k)}_\gamma$. They form the set of reduced large items $I^{\text{red}}_L := \bigcup_k \bigcup_\gamma \{a^{(k)}_\gamma\}$.

Similar to Lawler [20], we now prove that the overall solution quality does not decrease too much.

**Algorithm 1:** The algorithm to determine the $a^{(k)}_\gamma$.

```plaintext
for $k = 0, \ldots, \kappa$ do
    for $\gamma = 0, \ldots, 2^{\kappa+1}(\kappa + 1) - 1$ do
        $a^{(k)}_\gamma := \emptyset$;
    end
    $a^{(\kappa+1)}_0 := \emptyset$;
for $a \in I_L$ do
    Determine $(k(a), \gamma(a))$;
    if $s(a^{(k(a))}_\gamma) > s(a)$ or $a^{(k(a))}_\gamma = \emptyset$ then
        $a^{(k(a))}_\gamma := a$;
end
Output: $I^{\text{red}}_L := \bigcup_k \bigcup_\gamma \{a^{(k)}_\gamma\}$
```

**Lemma 7.** Let $0 \leq v \leq c = 1$. Then

$$OPT(\{a_{\text{eff}}\}, c-v) \geq OPT(I_S, c-v) - T$$

and

$$OPT(I^{\text{red}}_L, v) \geq \left(1 - \frac{1}{4 \log_2(\frac{1}{\varepsilon^2})} + 1 \varepsilon\right)OPT(I_L, v).$$

**Proof.** For the first inequality, there are two possibilities: either copies of $a_{\text{eff}}$ can be taken such that the entire capacity $c-v$ is used. Then obviously $OPT(\{a_{\text{eff}}\}, c-v) = OPT(I_S, c-v)$ holds. Otherwise, we have similar to the proof of Theorem 4 that $OPT(\{a_{\text{eff}}\}, c-v) + p(a_{\text{eff}}) = \left[\frac{c-v}{s(a_{\text{eff}})}\right] \cdot p(a_{\text{eff}}) \geq OPT(I_S, c-v)$. Thus, $OPT(\{a_{\text{eff}}\}, c-v) \geq OPT(I_S, c-v) - p(a_{\text{eff}}) \geq OPT(I_S, c-v) - T$. The first inequality follows.
For the second inequality, take an optimal solution \((x_a)\) such that \(OPT(I_L, v) = \sum_{a \in I_L} p(a)x_a\).
Replace now every item \(a\) by its counterpart \(a^{(k(a))}_{\gamma(a)}\) in \(I_{L, \text{red}}\). Obviously, the solution stays feasible, i.e. the volume \(v\) will not be exceeded, because an item may only be replaced by a smaller one. This solution has total profit \(\sum_{a \in I_L} p(a^{(k(a))}_{\gamma(a)})x_a\). Moreover, we have
\[
p(a^{(k(a))}_{\gamma(a)}) \geq p(a) - 2^{k(a)} K \sum_{j=1}^{2} p(a) - \frac{1}{\epsilon} \frac{1}{\log(2^{k(a)} - 1)} \frac{1}{2^{k(a)} - 1} \frac{1}{2^{k(a)} - 1} \cdot OPT(I_L, v) - \left(1 - \frac{1}{4\kappa + 1} \frac{1}{2^{k(a)} - 1}\right)
\]
by the definition of the \(L_{(k)}\). We get
\[
OPT(I_{L, \text{red}}, v) \geq \sum_{a \in I_L} p(a^{(k(a))}_{\gamma(a)})x_a \geq \sum_{a \in I_L} \left(1 - \frac{1}{4\kappa + 1} \frac{1}{2^{k(a)} - 1}\right) \cdot OPT(I_L, v)
\]
\[
= \left(1 - \frac{1}{4\kappa + 1} \frac{1}{2^{k(a)} - 1}\right) OPT(I_L, v).
\]

\(\square\)

**Theorem 8.** \(I_{L, \text{red}}\) has \(O\left(\frac{1}{\epsilon} \log^{2} \frac{1}{\epsilon}\right)\) many items. Algorithm 4 needs time \(O(n + \frac{1}{\epsilon} \log^{2} \frac{1}{\epsilon})\) and space in \(O\left(\frac{1}{\epsilon} \log^{2} \frac{1}{\epsilon}\right)\) for the construction and for saving \(I_{L, \text{red}}\).

**Proof.** Together with \(a^{(k+1)}_{\gamma}\), we have \(O((\kappa + 1) \cdot (2^{k+1}(\kappa + 1) - 1 + 1)) = O(\log \frac{1}{\epsilon} \cdot (\frac{1}{\epsilon} \log \frac{1}{\epsilon})) = O\left(\frac{1}{\epsilon} \log^{2} \frac{1}{\epsilon}\right)\) many items \(a^{(k)}_{\gamma}\). The space needed is asymptotically bounded by the space required to save the \(a^{(k)}_{\gamma}\). Finally, the running time is obviously bounded by \(O(n + \frac{1}{\epsilon} \log^{2} \frac{1}{\epsilon})\): the values \(k(a)\) and \(\gamma(a)\) can be found in \(O(1)\) because we assume that the logarithm can be determined in \(O(1)\). \(\square\)

**Remark 9.** If there is one item \(a\) with profit \(p(a) = 2P_0\), i.e. whose profit attains the upper bound, the optimum solution obviously consists of this single item. During the partition of \(I\) into \(I_L\) and \(I_S\), it can easily be checked whether such an item is part of \(I\). Since the algorithm can directly stop if this is the case, we will from now on assume without loss of generality that such an item does not exist and that \(a^{(k+1)}_0 = \emptyset\).

### 4 A Simplified Solution Structure

In this section, we will transform \(I_{L, \text{red}}\) into a new instance \(\tilde{I}\) whose optimum \(OPT(\tilde{I}, v)\) is only slightly smaller than \(OPT(I_{L, \text{red}}, v)\) and where the corresponding solution has a special structure. This new transformation will allow us later to faster construct the approximate solution. First, we define
\[
I^{(k)} := \left\{a \in I_{L, \text{red}} \mid p(a) \in L^{(k)}\right\} = \left\{a \in I_{L, \text{red}} \mid p(a) \in [2^{kT}, 2^{k+1}T]\right\}.
\]
Note that the items are already partitioned into the \(I^{(k)}\) because of the construction of \(I_{L, \text{red}}\).
**Theorem 13.** It is additionally proved that at most one item of every $OPT$ is needed. First, we introduce a definition for the proof.

**Definition 10.** Let $a_1, a_2$ be two knapsack items with $s(a_1) + s(a_2) \leq c$. The gluing operation $\oplus$ combines them into a new item $a_1 \oplus a_2$ with $p(a_1 \oplus a_2) = p(a_1) + p(a_2)$ and $s(a_1 \oplus a_2) = s(a_1) + s(a_2)$.

Thus, the gluing operation is only defined on pairs of items whose combined size does not exceed $c$.

The basic idea for the new instance $\tilde{I}$ is as follows: we first set $\tilde{I}(0) := I(0)$. Then, we construct $a_1 \oplus a_2$ for all $a_1, a_2 \in \tilde{I}(0)$ (which also includes the case $a_1 = a_2$), which forms the item set $\tilde{I}(1) := \{a_1 \oplus a_2 | a_1, a_2 \in \tilde{I}(0)\}$. Note that $p(a_1 \oplus a_2) \in [2T, 4T] = L(1)$. For every profit interval $L(1)$, we keep only the item of smallest size in $I(1) \cup \tilde{I}(1)$, which yields the item set $\tilde{I}(1)$. This procedure is iterated for $k = 1, \ldots, \kappa - 1$: the set $\tilde{I}(k)$ contains the items with profit in $[2^k T, 2^{k+1} T) = L(k)$ (see Fig. 1(a)). Gluing like above yields the item set $\tilde{I}(k+1)$ with profits in $[2^{k+1} T, 2^{k+2} T) = L(k+1)$ (see Fig. 1(b)). By taking again the smallest item in $\tilde{I}(k+1) \cup I(k-1)$ for every $L(k+1)$, the set $\tilde{I}(k+1)$ is derived (see Fig. 1(c)). The item in $\tilde{I}(k+1)$ with profit in $L(k+1)$ is denoted by $\tilde{a}_\gamma(k+1)$.

We finish when $\tilde{I}(\kappa)$ has been constructed: we are in the case where $I(\kappa+1) = \emptyset$, i.e. $a_0(\kappa+1) = \emptyset$, so that it is not necessary to construct $\tilde{I}(\kappa+1)$. Hence, we also have $\tilde{a}_0(\kappa+1) = \emptyset$.

Note that we may glue items together that already consist of glued items. For backtracking, we save for every $\tilde{a}_\gamma(k)$ which two items in $\tilde{I}(k-1)$ have formed it or whether $\tilde{a}_\gamma(k)$ has already been an item in $I(k)$. Algorithm 2 presents one way to construct the sets $\tilde{I}(k)$.

**Remark 11.** One item $\tilde{a}_\gamma(k)$ is in fact the combination of several items in $L_{\text{red}}$, the profit and size of $\tilde{a}_\gamma(k)$ is equal to the total profit and size of these items. The $\tilde{a}_\gamma(k)$ represent feasible item combinations because an arbitrary number of item copies can be taken in UKP.

The item set

$$\tilde{I} := \bigcup_{k=0}^{\kappa} \tilde{I}(k)$$

has for every $0 \leq v \leq c$ a solution near the original optimum $OPT(I_{L,\text{red}}, v)$ as shown below in Theorem 13. It is additionally proved that at most one item of every $\tilde{I}(k)$ for $k \in \{0, \ldots, \kappa - 1\}$ is needed. First, we introduce a definition for the proof.

**Definition 12.** Let $I'$ be a set of knapsack items with $p(a) \geq T$ for every $a \in I'$. We denote by $OPT_{\leq k_0}(I', v)$ the optimum profit to the fill knapsack volume $v \leq c$ with the items in $I'$ where for every $k \in \{0, \ldots, k_0\}$ at most one item with profit in $L(k)$ is used.

For instance,

$$OPT_{\leq k_0} \left( \tilde{I}(0) \cup \ldots \cup \tilde{I}(k_0) \cup \tilde{I}(k_0+1) \cup \ldots \cup \tilde{I}(\kappa), v \right)$$

uses only one item from every $\tilde{I}(k)$ for $k \in \{0, \ldots, k_0\}$.

**Theorem 13.** For $v \leq c$ and $k_0 = 0, \ldots, \kappa - 1$, we have

$$OPT_{\leq k_0} \left( \bigcup_{k=0}^{k_0+1} \tilde{I}(k) \cup \bigcup_{k=k_0+2}^{\kappa} I(k), v \right) \geq \left( 1 - \frac{\varepsilon}{4 \log_2 \left( \frac{2}{\varepsilon} \right) + 1} \right)^{k_0+1} OPT(I_{L,\text{red}}, v).$$
(a) The items in $I^{(k)}$ and $I^{(k+1)}$. The height of every item $a$ corresponds to its size $s(a)$ while its position on the axis corresponds to its profit $p(a)$. The axis is partitioned into the profit intervals $L^{(k)} = [2^k T + \gamma \cdot 2^k K, 2^k T + (\gamma + 1)2^k K]$.

(b) The set $I^{(k+1)}$ together with the newly constructed items in $\tilde{I}^{(k+1)}$.

(c) The new set $\tilde{I}^{(k+1)}$ after keeping only the smallest item with profit in $L^{(k+1)}$. For instance, $\tilde{a}_4 \oplus \tilde{a}_4$ is kept because it is the smallest item in its profit interval $L^{(k+1)}$.

Figure 1: Principle of deriving $\tilde{I}^{(k+1)}$ from $\tilde{I}^{(k)}$ and $I^{(k+1)}$.
**Algorithm 2:** The construction of the item sets \( I^{(k)} \).

\[
\begin{array}{l}
\text{for } k = 0, \ldots, \kappa \text{ do} \\
\quad \text{for } \gamma = 0, \ldots, 2^{\kappa+1} (\kappa + 1) - 1 \text{ do} \\
\quad\quad a^{(k)}_{\gamma} := a^{(k)}_{\gamma}; \\
\quad\quad \text{Backtrack}(a^{(k)}_{\gamma}) := a^{(k)}_{\gamma}; \\
\quad \tilde{I}^{(0)} := I^{(0)}; \\
\text{for } k = 0, \ldots, \kappa - 1 \text{ do} \\
\quad \text{for } \gamma = 0, \ldots, 2^{\kappa+1} (\kappa + 1) - 1 \text{ do} \\
\quad\quad \text{if } s(\tilde{a}^{(k)}_{\gamma}) + s(\tilde{a}^{(k)}_{\gamma'}) \leq c \text{ then} \\
\quad\quad\quad \tilde{a} := \tilde{a}^{(k)}_{\gamma} \oplus \tilde{a}^{(k)}_{\gamma'}; \\
\quad\quad\quad \text{if } s(\tilde{a}) < s(\tilde{a}^{(k+1)}_{\gamma}) \text{ or } \tilde{a}^{(k+1)}_{\gamma} = \emptyset \text{ then} \\
\quad\quad\quad\quad \tilde{a}^{(k+1)}_{\gamma} := \tilde{a}; \\
\quad\quad\quad\quad \text{Backtrack}(\tilde{a}^{(k+1)}_{\gamma}) := (\tilde{a}^{(k)}_{\gamma}, \tilde{a}^{(k')}_{\gamma'}); \\
\quad \tilde{I}^{(k+1)} := \{\tilde{a}^{(k+1)}_{0}, \ldots, \tilde{a}^{(k+1)}_{2^{\kappa+1}(\kappa+1)-1}\}; \\
\end{array}
\]

**Proof.** The proof idea is quite simple: we iteratively replace the items in \( I^{(k_0+1)} \) by their counterpart in \( I^{(k_0+1)} \) and replace every pair of items in \( I^{(k_0)} \) also by the counterpart in \( I^{(k_0+1)} \). This directly follows the way to construct the item sets \( I^{(k)} \) presented in Algorithm 2.

Formally, the statement is proved by induction over \( k_0 \). Let \( k_0 = 0 \). Take an optimum solution for \( \tilde{I}^{(0)} \cup I^{(1)} \cup \ldots \cup I^{(\kappa)} = I^{(0)} \cup I^{(1)} \cup \ldots \cup I^{(\kappa)} = I_{L,\text{red}} \). For ease of notation, we directly write each item as often as it appears in the solution. We have three sub-sequences:

- Let \( \tilde{a}_1, \ldots, \tilde{a}_{\eta} (\eta \in \mathbb{N}) \) be the items from \( \tilde{I}^{(0)} = I^{(0)} \) in the optimal solution for \( OPT(I_{L,\text{red}}, v) \).
  We suppose that \( \eta \) is odd (the case where \( \eta \) is even is easier and handled below.)
- Let \( \tilde{a}_{\eta+1}, \ldots, \tilde{a}_{\eta+\xi} (\xi \in \mathbb{N}) \) be the items from \( I^{(1)} \) in the optimal solution for \( OPT(I_{L,\text{red}}, v) \).
- Let \( \tilde{a}'_1, \ldots, \tilde{a}'_{\lambda} (\lambda \in \mathbb{N}) \) be the remaining items from \( I^{(2)} \cup \ldots \cup I^{(\kappa)} \) in the optimal solution for \( OPT(I_{L,\text{red}}, v) \). This set is denoted by \( \Lambda \). As defined above, the total profit of these items is written as \( p(\Lambda) \).

See Figure 2[a] for an illustration. We have

\[
OPT \left( \tilde{I}^{(0)} \cup I^{(1)} \cup \ldots \cup I^{(\kappa)}, v \right) = \sum_{i=1}^{\eta} p(\tilde{a}_i) + \sum_{j=\eta+1}^{\eta+\xi} p(\tilde{a}_j) + p(\Lambda). \tag{6}
\]

In the first step, every two items \( \bar{a}_{2i-1} \) and \( \bar{a}_{2i} \) from \( \tilde{I}^{(0)} \) for \( i = 1, \ldots, \lfloor \frac{\eta}{2} \rfloor \) are replaced by \( \bar{a}_{2i-1} \oplus \bar{a}_{2i} \in \tilde{I}^{(1)} \) (see Fig. 2[b]). In the second step, every item \( \bar{a}_{2i-1} \oplus \bar{a}_{2i} \) is again replaced by the corresponding item \( \tilde{a}^{(1)}_{\gamma(\bar{a}_{2i-1} \oplus \bar{a}_{2i})} =: \tilde{a}^{(1)}_{\rho(i)} \) in \( \tilde{I}^{(1)} \) (for \( i = 1, \ldots, \lfloor \frac{\eta}{2} \rfloor \)). Only item \( \tilde{a}_\eta \) remains unchanged.
Moreover, $\bar{a}_j$ from $I^{(l)}$ is replaced by the corresponding $\tilde{a}_{\gamma_j(\bar{a}_j)} =: \tilde{a}_{\rho_j(j)}^{(1)}$ for $j = \eta + 1, \ldots, \eta + \xi$ (see Fig. 2 (c)). Note that this new solution is indeed feasible because the replacing items $\tilde{a}_{\gamma_j}^{(1)}$ are at most as large as the original ones. Moreover, the corresponding items $\tilde{a}_{\rho_j(i)}^{(1)}$ and $\tilde{a}_{\rho_j(j)}^{(1)}$ must exist by the construction of $\tilde{I}^{(1)}$. Thus, we have a (feasible) solution that consists of only one item in $\tilde{I}^{(0)}$, the items $\tilde{a}_{\gamma_j}^{(1)}$ and $\tilde{a}_{\gamma_j(\bar{a}_j)}^{(1)}$ in $\tilde{I}^{(1)}$ and the remaining items $\tilde{a}_1', \ldots, \tilde{a}_{\lambda}$ in $I^{(2)}, \ldots, I^{(\kappa)}$: this solution respects the structure of $OPT_{\leq k}(\cdot, v)$ (If $\eta$ is even, no item in $\tilde{I}^{(0)}$ is used.)

Let now $\tilde{a}$ be an item $\tilde{a}_{2i-1} \oplus \tilde{a}_{2i}$ or $\tilde{a}_j$. It can be proved like for Inequality (5) that

$$p(\tilde{a}_j^{(1)}) \geq \left(1 - \frac{\varepsilon}{4 \log_2(\frac{2}{\varepsilon}) + 1}\right) p(\tilde{a}) .$$

Thus, we have

$$OPT_{\leq 0} \left(\tilde{I}^{(0)} \cup \tilde{I}^{(1)} \cup I^{(2)} \cup \ldots \cup I^{(\kappa)}, v\right) \geq p(\tilde{a}) + \sum_{j=1}^{\eta} p(\tilde{a}_j^{(1)}) + \sum_{j=\eta+1}^{\eta+\xi} p(\tilde{a}_j^{(1)}) + p(\Lambda)$$

$$\geq p(\tilde{a}) + \left(1 - \frac{\varepsilon}{4 \log_2(\frac{2}{\varepsilon}) + 1}\right) \sum_{j=1}^{\eta} p(\tilde{a}_{2i-1} \oplus \tilde{a}_{2i})$$

$$+ \left(1 - \frac{\varepsilon}{4 \log_2(\frac{2}{\varepsilon}) + 1}\right) \sum_{j=\eta+1}^{\eta+\xi} p(\tilde{a}_j) + p(\Lambda)$$

$$\geq \left(1 - \frac{\varepsilon}{4 \log_2(\frac{2}{\varepsilon}) + 1}\right) \left(\sum_{i=1}^{\eta} p(\tilde{a}_i) + \sum_{j=\eta+1}^{\eta+\xi} p(\tilde{a}_j) + p(\Lambda)\right)$$

$$\geq \left(1 - \frac{\varepsilon}{4 \log_2(\frac{2}{\varepsilon}) + 1}\right) OPT \left(\tilde{I}^{(0)} \cup \tilde{I}^{(1)} \cup I^{(2)} \cup \ldots \cup I^{(\kappa)}, v\right)$$

$$= \left(1 - \frac{\varepsilon}{4 \log_2(\frac{2}{\varepsilon}) + 1}\right) OPT \left(I_{L,\text{red}}, v\right) .$$

The statement for $k_0 = 1, \ldots, \kappa - 1$ now follows by induction. The proof is almost identical to the case $k_0 = 0$ above, the only difference is that there are additionally the items in $\tilde{I}^{(0)}, \ldots, \tilde{I}^{(k_0-1)}$ that remain unchanged like the items $I^{(k_0+2)}, \ldots, I^{(\kappa)}$. Only the items in $\tilde{I}^{(k_0)}$ and $I^{(k_0)}$ are replaced (with the possible exception of one item in $\tilde{I}^{(k_0)}$).

**Lemma 14.** We have $OPT(\tilde{I} \cup \{a_{\text{eff}}\}) \leq OPT(I_{L,\text{red}} \cup I_S) \leq OPT(I_{L} \cup I_S) = OPT(I) \leq 2P_0$.

**Proof.** $\tilde{I}$ consists of items in $I_{L,\text{red}}$ or of items that can be obtained by gluing several items in $I_{L,\text{red}}$ together. Every combination of items in $\tilde{I}$ can therefore be represented by items in $I_{L,\text{red}}$. Moreover, we have $a_{\text{eff}} \in I_S$. The first inequality follows. Since $I_{L,\text{red}} \subseteq I_L$, the second inequality is obvious.

Up to now, we have reduced the original item set $I$ to $\tilde{I} \cup \{a_{\text{eff}}\}$. 

---

11
(a) The current solution for $\tilde{\mathcal{I}}(0) \cup \cdots \cup \tilde{\mathcal{I}}(k) \cup I(k+1) \cup \cdots \cup I(\kappa)$. The structure of $OPT_{\leq k-1}(\cdot, v)$ is respected, i.e. at most one item from every $\tilde{\mathcal{I}}(0), \ldots, \tilde{\mathcal{I}}(k-1)$ is used.

(b) The items in $\tilde{\mathcal{I}}(k)$ are pairwise glued together with the possible exception of one item.

(c) The items in $\tilde{\mathcal{I}}(k+1) \cup I(k+1)$ are replaced by their counterpart in $I(k+1)$. Now, at most one item in $\tilde{\mathcal{I}}(k)$ is part of the solution, and the structure for $OPT_{\leq k}(\cdot, v)$ is respected.

Figure 2: The principle of the proof for Theorem 13
Lemma 15. Assume as mentioned in Remark 9 that $a^{(k+1)} = \emptyset$. Consider the solutions for $\tilde{I} \cup \{a_{\text{eff}}\}$ where the items in $\tilde{I}$ respect the structure of Definition 12 for $k_0 = \kappa - 1$. This means that at most one item is used from every $\tilde{I}^{(k)}$ for $k \in \{0, \ldots, \kappa - 1\}$. ($a_{\text{eff}}$ has profit $p(a_{\text{eff}}) < T$ such that it does not have to satisfy any structure conditions.) Then there are two possible cases:

- One solution uses (at least) two items in $\tilde{I}^{(\kappa)}$. This is the case if and only if the optimum for $\tilde{I} \cup \{a_{\text{eff}}\}$ is $2P_0$ and consists of two item copies of the item $a^{(\kappa)}_0$ with $p(a^{(\kappa)}_0) = P_0$.

- Every solution uses at most one item in $\tilde{I}^{(\kappa)}$. Then, $OPT_{\leq \kappa-1}(\tilde{I}, v') = OPT_{\leq \kappa}(\tilde{I}, v')$ holds for all values $0 \leq v' \leq c$, and there is a value $0 \leq v \leq c$ such that

$$OPT_{\leq \kappa}(\tilde{I}, v) + OPT(\{a_{\text{eff}}\}, c - v) = OPT_{\leq \kappa-1}(\tilde{I}, v) + OPT(\{a_{\text{eff}}\}, c - v) \geq \left(1 - \frac{\varepsilon}{4 \log_2(\frac{\varepsilon}{\kappa}) + 1}\right)^{\kappa+1} OPT(\tilde{I}) - T.$$

In the second case, $OPT_{\leq \kappa}(\tilde{I}, v)$ uses at least one item in $\tilde{I}^{(\kappa-2)} \cup \tilde{I}^{(\kappa-1)} \cup \tilde{I}^{(\kappa)}$, and/or we have $OPT(\{a_{\text{eff}}\}, c - v) \geq \frac{1}{2}P_0$.

Proof. Note that $I_{\text{L,red}}$ does not contain any item with profit $2P_0$ (see Remark 9). By construction, this is still the case for $\tilde{I}$. Suppose now that one solution for $\tilde{I} \cup \{a_{\text{eff}}\}$ uses more than one item in $\tilde{I}^{(\kappa)}$. Since items in $\tilde{I}^{(\kappa)}$ have profit in $|P_0, 2P_0|$, only two copies of item $a_0^{(\kappa)}$—here with $p(a_0^{(\kappa)}) = P_0$—can be used. In fact, $2P_0$ is the maximum possible profit because $OPT(\tilde{I} \cup \{a_{\text{eff}}\}) \leq OPT(\tilde{I}) \leq 2P_0$ as we have seen in Lemma 14. Thus, the “only if” direction has been proved. The “if”-direction is obvious.

Suppose now that every solution for $\tilde{I} \cup \{a_{\text{eff}}\}$ that respects the structure for $k_0 = \kappa - 1$ uses at most one item in $\tilde{I}^{(\kappa)}$. Thus, $OPT_{\leq \kappa-1}(\tilde{I}, v') = OPT_{\leq \kappa}(\tilde{I}, v')$ holds for all $0 \leq v' \leq c$.

Let $v \leq c$ now be the volume the large items $I_L$ occupy in an optimum solution for $\tilde{I}$. Then obviously $OPT(\tilde{I}) = OPT(\tilde{I}_L, v) + OPT(I_S, c - v)$ holds. We have the following inequality:

$$OPT_{\leq \kappa-1}(\tilde{I}, v) + OPT(\{a_{\text{eff}}\}, c - v) = OPT_{\leq \kappa}(\tilde{I}, v) + OPT(\{a_{\text{eff}}\}, c - v) \geq \left(1 - \frac{\varepsilon}{4 \log_2(\frac{\varepsilon}{\kappa}) + 1}\right)^{\kappa+1} OPT(I_{\text{L,red}}, v) + OPT(\{a_{\text{eff}}\}, c - v) \geq \left(1 - \frac{\varepsilon}{4 \log_2(\frac{\varepsilon}{\kappa}) + 1}\right)^{\kappa+1} OPT(I_L, v) + OPT(I_S, c - v) - T \geq \left(1 - \frac{\varepsilon}{4 \log_2(\frac{\varepsilon}{\kappa}) + 1}\right)^{\kappa+1} OPT(I) - T.$$

For the final property, suppose that no item in $\tilde{I}^{(\kappa-2)} \cup \tilde{I}^{(\kappa-1)} \cup \tilde{I}^{(\kappa)}$ is used in a solution for $OPT_{\leq \kappa}(\tilde{I}, v)$. Then we have

$$OPT_{\leq \kappa}(\tilde{I}, v) \leq \sum_{k=0}^{\kappa-3} \max \left\{ p(a) \mid a \in \tilde{I}^{(k)} \right\} \leq \sum_{k=0}^{\kappa-3} 2 \cdot 2^k T < 2^{\kappa-1}T \frac{1}{2} < \frac{1}{2}P_0.$$
On the other hand, Inequality \[8\] together with \((1 - \delta)^k \geq (1 - k \cdot \delta)\) for \(\delta < 1\) yields

\[
OPT_{\leq \kappa} \left( \tilde{I}, v \right) + OPT \left( \{ a_{\text{eff}} \}, c - v \right) \geq \left( 1 - \frac{\varepsilon}{4} \right) OPT(I) - T
\]

\[
\left( 1 - \frac{\varepsilon}{4} \right) OPT(I) - \frac{1}{2} \varepsilon P_0 \geq \left( 1 - \frac{\varepsilon}{4} \right) OPT(I) - \frac{1}{2} \varepsilon OPT(I)
\]

\[
\varepsilon \leq 1/4, \quad \frac{3}{4} OPT(I) \geq \frac{3}{4} P_0.
\]

Hence, \(OPT(\{ a_{\text{eff}} \}, c - v) \geq \frac{1}{4} P_0\) holds. The final property of the second case follows. \(\square\)

**Definition 16.** Take \(\left\lfloor \frac{P_0}{\varepsilon OPT(a_{\text{eff}})} \right\rfloor\) items \(a_{\text{eff}}\). If their total size is smaller than \(c\), they are glued together to \(a_{\text{eff} - c}\).

Obviously, \(a_{\text{eff} - c}\) consists of the smallest number of items \(a_{\text{eff}}\) whose total profit is at least \(\frac{P_0}{4}\), and \(a_{\text{eff} - c}\) is a large item.

**Definition 17.** Let \(OPT_{\text{St}}(\tilde{I} \cup \{ a_{\text{eff} - c} \}, v)\) be the optimal solution for \(v \leq c\) that satisfies the following conditions:

1. Similar to \(OPT_{\leq \kappa}(\cdot, v)\), it uses for every \(k \in \{0, \ldots, \kappa\}\) at most one item in \(\tilde{I}(k)\) and also item \(a_{\text{eff} - c}\) at most once.

2. It uses at least one item \(\tilde{a} \in \tilde{I}(\kappa - 2) \cup \tilde{I}(\kappa - 1) \cup \tilde{I}(\kappa) \cup \{ a_{\text{eff} - c} \}\).

Such solutions therefore have profit at least \(p(\tilde{a}) \geq \frac{2^{\kappa - 2}T}{4} = \frac{1}{4} P_0\). If \(v\) is too small such that such a solution does not exist, we set \(OPT_{\text{St}}(\tilde{I} \cup \{ a_{\text{eff} - c} \}, v) = 0\).

**Theorem 18.** In the second case of Lemma 15, there is a value \(0 \leq v \leq c\) such that

\[
OPT_{\text{St}} \left( \tilde{I} \cup \{ a_{\text{eff} - c} \}, v \right) + OPT \left( \{ a_{\text{eff}} \}, c - v \right) \geq \left( 1 - \frac{\varepsilon}{4} \right) \kappa \log_2 \left( \frac{2}{\varepsilon} \right) + 1 \quad OPT(I) - T.
\]

**Proof.** Like in the proof of Lemma 15, let \(v'\) be the volume the large items \(I_L\) occupy in an optimum solution for \(I\). Consider an optimum solution for \(OPT_{\leq \kappa}(\tilde{I}, v')\) and suppose that it does not use any item in \(\tilde{I}(\kappa - 2) \cup \tilde{I}(\kappa - 1) \cup \tilde{I}(\kappa)\). Lemma 15 states that \(OPT(\{ a_{\text{eff}} \}, c - v')\) has profit at least \(\frac{1}{4} P_0\). Thus, a subset of them can be replaced by \(a_{\text{eff} - c}\), and \(c - v' \geq s(a_{\text{eff} - c})\). We set \(v := v' + s(a_{\text{eff} - c})\). Note that \(OPT_{\text{St}}(\tilde{I} \cup \{ a_{\text{eff} - c} \}, v) \geq OPT_{\leq \kappa}(\tilde{I}, v') + p(a_{\text{eff} - c})\). Moreover, \(OPT_{\leq \kappa - 1}(\tilde{I}, v') = OPT_{\leq \kappa}(\tilde{I}, v')\).
holds because we are in the second case of Lemma 15. We get the following inequality:

\[
\text{OPT}_\text{St} \left( \tilde{I} \cup \{a_{\text{eff} - c}\}, v \right) + \text{OPT} \left( \{a_{\text{eff}}\}, c - v \right) \\
\geq \text{OPT}_{\leq \kappa} \left( \tilde{I}, v' \right) + p(a_{\text{eff} - c}) + \text{OPT} \left( \{a_{\text{eff}}\}, c - v' - s(a_{\text{eff} - c}) \right) \\
= \text{OPT}_{\leq \kappa - 1} \left( \tilde{I}, v' \right) + \text{OPT} \left( \{a_{\text{eff}}\}, c - v' \right)
\]

Thm. 13 \quad \geq \left( 1 - \frac{\varepsilon}{4 \log_2 \left( \frac{2}{\varepsilon} \right) + 1} \right)^\kappa \text{OPT} \left( I_{L, \text{red}}, v' \right) + \text{OPT} \left( \{a_{\text{eff}}\}, c - v' \right)

Lem. 17 \quad \geq \left( 1 - \frac{\varepsilon}{4 \log_2 \left( \frac{2}{\varepsilon} \right) + 1} \right)^{\kappa + 1} \text{OPT} \left( I_L, v' \right) + \text{OPT} \left( I_S, c - v' \right) - T

Kram. 17 \quad \geq \left( 1 - \frac{\varepsilon}{4 \log_2 \left( \frac{2}{\varepsilon} \right) + 1} \right)^{\kappa + 1} \left( \text{OPT} \left( I_L, v' \right) + \text{OPT} \left( I_S, c - v' \right) \right) - T

Kram. 17 \quad = \left( 1 - \frac{\varepsilon}{4 \log_2 \left( \frac{2}{\varepsilon} \right) + 1} \right)^{\kappa + 1} \text{OPT}(I) - T.

Note that \text{OPT}(\{a_{\text{eff}}\}, c - v) is well-defined—and therefore the entire chain of inequalities feasible—because \( c - v = c - v' - s(a_{\text{eff} - c}) \geq 0 \).

Suppose now that the optimal solution uses at least one item in \( \tilde{I}^{(\kappa - 2)} \cup \tilde{I}^{(\kappa - 1)} \cup \tilde{I}^{(\kappa)} \). We can then directly set \( v := v' \), and the proof is similar to the first case.

Roughly speaking, a solution in the first case satisfies the estimate of the theorem and uses at most one item in every \( \tilde{I}^{(\kappa)} \), but no item in \( \tilde{I}^{(\kappa - 2)}, \tilde{I}^{(\kappa - 1)} \) or \( \tilde{I}^{(\kappa)} \). This implies that enough items \( a_{\text{eff}} \) are part of the solution such that a subset of them can be replaced by \( a_{\text{eff} - c} \).

So far, we have not constructed an actual solution. We only have showed in Theorem 18 that a solution for \( \tilde{I} \cup \{a_{\text{eff} - c}\} \cup \{a_{\text{eff}}\} \) that is close to \text{OPT}(I) and that satisfies the structure of Definition 17 exists.

**Theorem 19.** The cardinality of \( \tilde{I}^{(k)} \) is in \( O \left( \frac{1}{\varepsilon} \log \frac{1}{\varepsilon} \right) \), i.e. \( \tilde{I} \) has \( O \left( \frac{1}{\varepsilon} \log^2 \frac{1}{\varepsilon} \right) \) many items. Algorithm 2 constructs \( \tilde{I} \) in time \( O \left( \frac{1}{\varepsilon} \log^3 \left( \frac{1}{\varepsilon} \right) \right) \) and space \( O \left( \frac{1}{\varepsilon} \log^2 \frac{1}{\varepsilon} \right) \), which also includes the space to save \( \tilde{I} \) and the backtracking information. The item \( a_{\text{eff} - c} \) can be constructed in \( O(1) \).

**Proof.** The statement for \( a_{\text{eff} - c} \) is trivial: the number of items \( a_{\text{eff} - c} \) to glue together can be determined by division.

The number of items in \( \tilde{I}^{(k)} \) and \( \tilde{I} \) can be derived like the number of items in \( I_{L, \text{red}} \) in Theorem 8. The running time of Algorithm 2 is obviously dominated by the second for-loop. It is in

\[
O \left( \kappa \cdot \left( 2^{\kappa + 1} (\kappa + 1) \right)^2 \right) = O \left( \log \left( \frac{1}{\varepsilon} \right) \cdot \left( \frac{1}{\varepsilon} \log \frac{1}{\varepsilon} \right)^2 \right) = O \left( \frac{1}{\varepsilon^2} \log^3 \left( \frac{1}{\varepsilon} \right) \right).
\]

The space complexity is dominated by the space to save the \( a_{\gamma}^{(k)} \) and the backtracking information, which is again asymptotically equal to the number of items in \( \tilde{I} \).
5 Finding an Approximate Structured Solution by Dynamic Programming

The previous section has presented three cases:

1. If \( I \) has one item of profit \( 2P_0 \): return this item, and \( OPT(I) = 2P_0 \) (see Remark 9).

2. If this is not the case, and \( \tilde{I} \) has one item of profit \( P_0 \) and size at most \( \frac{s}{2} \), two copies of this item are the optimum solution for \( \tilde{I} \cup \{a_{eff}\} \) (see Lemma 15). Undoing the gluing returns the optimum solution with \( OPT(I) = 2P_0 \).

3. Otherwise, there is an approximate solution to \( \tilde{I} \cup \{a_{eff-c}\} \cup \{a_{eff}\} \) where the large items respect the structure of Definition 17, at most one item from every \( \tilde{I}^{(k)} \) for \( k \in \{0, \ldots, \kappa\} \) is used and at least one item \( \tilde{a} \in \tilde{I}^{(\kappa-2)} \cup \tilde{I}^{(\kappa-1)} \cup \tilde{I}^{(\kappa)} \cup \{a_{eff-c}\} \).

The first two cases can be easily checked such that the main difficulty is the last case. Therefore, we will from now on assume that every structured solution for \( \tilde{I} \cup \{a_{eff-c}\} \cup \{a_{eff}\} \) satisfies the properties of the third case.

We use dynamic programming to find for all \( 0 \leq v \leq c \) the corresponding set of large items \( V \subseteq \tilde{I} \cup \{a_{eff-c}\} \) with \( s(V) \leq v \). For convenience, let \( \tilde{I}^{(\kappa+1)} := \{a_{eff-c}\} \). Similar to Lawler [20], we introduce tuples \((p, s, k)\). For profit \( p \) with \( 0 \leq p \leq 2P_0 \) and size \( 0 \leq s \leq c \), the tuple \((p, s, k)\) states that there is an item set of size \( s \) whose total profit is \( p \). Moreover, the set has only items in \( \tilde{I}^{(k)} \cup \ldots \cup \tilde{I}^{(\kappa+1)} \) and respects the structure above.

The dynamic program is quite simple: start with the dummy tuple set \( F^{(\kappa+2)} := \{(0, 0, \kappa + 2)\} \). For \( k = \kappa + 1, \ldots, \kappa - 2 \), the tuples in \( F^{(k)} \) are constructed by

\[
F^{(k)} := \left\{ (p, s, k) \mid (p, s, k + 1) \in F^{(k+1)} \right\} 
\cup \left\{ (p + p(\tilde{a}), s + s(\tilde{a}), k) \mid (p, s, k + 1) \in F^{(k+1)}, \tilde{a} \in \tilde{I}^{(k)}, s + s(\tilde{a}) \leq c \right\} .
\]

Note that \((0, 0, k + 1)\) is in \( F^{(k+1)} \), which guarantees that \( F^{(k)} \) also contains the entries \((p(\tilde{a}), s(\tilde{a}), k)\) for \( \tilde{a} \in \tilde{I}^{(k)} \) for \( k = \kappa + 1, \ldots, \kappa - 2 \). For \( k = \kappa - 3, \ldots, 0 \), this tuple \((0, 0, k + 1)\) is no longer considered to form the new tuples, which guarantees that tuples of the form \((p + p(\tilde{a}), s + s(\tilde{a}), k)\) for \( \tilde{a} \in \tilde{I}^{(k)} \) have \( p, s \neq 0 \). The recursion becomes

\[
F^{(k)} := \left\{ (p, s, k) \mid (p, s, k + 1) \in F^{(k+1)} \right\} 
\cup \left\{ (p + p(\tilde{a}), s + s(\tilde{a}), k) \mid (p, s, k + 1) \in F^{(k+1)} \setminus \{(0, 0, k + 1)\}, \tilde{a} \in \tilde{I}^{(k)}, s + s(\tilde{a}) \leq c \right\} .
\]

The actual item set corresponding to \((p, s, k)\) can be reconstructed by saving every time a new tuple is formed from which pair of item and tuple it was derived, which allows for backtracking.

**Definition 20.** A tuple \((p_2, s_2, k)\) is dominated by \((p_1, s_2, k)\) if \( p_2 \leq p_1 \) and \( s_2 \geq s_1 \).

Like in the paper by Lawler [20], dominated tuples \((p, s, k + 1)\) are now removed from \( F^{(k+1)} \) before \( F^{(k)} \) is constructed. This does not affect the outcome: dominated tuples only stand for sets of items with a profit not larger and a size not smaller than non-dominated tuples. A non-dominated tuple \((p, s, k)\) is therefore optimal, i.e. profit \( p \) can only be obtained with items of size at least \( s \) if items in \( \tilde{I}^{(k)}, \ldots, \tilde{I}^{(\kappa+1)} \) are considered.
Lemma 21. A tuple $(p, s, k) \in F^{(k)}$ stands for an item set satisfying the structure property of Definition 17. Therefore, we have $p \geq 2^{\kappa-2}T$ if $p > 0$. For every $v \leq c$, there is a tuple $(p, s, 0) \in F^{(0)}$ with $p = OPT_{St}(\tilde{I} \cup \{a_{\text{eff}, c}\}, v)$ and $s \leq v$.

Proof. This lemma directly follows from the dynamic program: tuples use at most one item from every $\tilde{I}^{(k)}$. For $k \in \{\kappa - 2, \ldots, \kappa + 1\}$, a tuple with $p > 0$ represents an item set that uses at least one item in $\tilde{I}^{(k)}, \ldots, \tilde{I}^{(k+1)}$. Tuples for $k \leq \kappa - 3$ are only derived from tuples that use at least one item in $\tilde{I}^{(\kappa-2)}, \ldots, \tilde{I}^{(\kappa+1)}$. If dominated sets are not removed, the dynamic program obviously constructs tuples for all possible combinations of items that satisfy the structure property, especially the optimum combinations for every $0 \leq v \leq c$. Removing dominated tuples does not affect the tuples that stand for the optimum item combinations so that the second property still holds.

While the dynamic programming above constructs the desired tuples, their number may increase dramatically until $F^{(0)}$ is obtained. We therefore use approximate dynamic programming for the tuples with profit in $[\frac{1}{4}P_0, 2P_0]$. This method is inspired by the dynamic programming used in [15] (see also [18] pp. 97–112).

Definition 17 and Lemma 21 state that a tuple $(p, s, k)$ with $p > 0$ satisfies $p \geq 2^{\kappa-2}T$. Apart from $(0, 0, k)$, all tuples have therefore profits in the interval $[2^{\kappa-2}T, 2P_0]$. We partition this interval into sub-intervals of length $2^{\kappa-2}K$. We get

$$[2^{\kappa-2}T, 2P_0] = \bigcup_{\xi=0}^{\xi_0} \left[ 2^{\kappa-2}T + \xi \cdot 2^{\kappa-2}K, 2^{\kappa-2}T + (\xi + 1)2^{\kappa-2}K \right] \cup \{2P_0\}$$

$$=: \bigcup_{\xi=0}^{\xi_0} \tilde{I}^{(\kappa-2)}_\xi \cup \tilde{L}^{(\kappa-2)}_{\xi_0+1}$$

for $\xi_0 := 7(\kappa + 1)2^{\kappa+1} - 1$. (A short calculation shows that $2^{\kappa-2}T + (\xi_0 + 1)2^{\kappa-2}K = 2P_0$.) The approximate dynamic programming keeps only the tuple $(p, s, k)$ for $p \in \tilde{I}^{(\kappa-2)}_\xi$ that has the smallest size $s$, and does so for all $\xi = 0, \ldots, \xi_0 + 1$. The modified dynamic program is presented in Algorithm 3 and shown in Figure 3. The sets of tuples are denoted by $D^{(k)}$. For convenience, let $(p(\xi), s(\xi), k) \in D^{(k)}$ be the smallest tuple with profit in $\tilde{I}^{(\kappa-2)}_\xi$. We again save the backtracking information during the execution of the algorithm.

Lemma 22. Let $\tilde{D}^{(k)}$ be the tuples from Algorithm 3 before the dominated entries are removed to get $D^{(k)}$. A tuple $(p, s, k) \in \tilde{D}^{(k)}$ for $k = \kappa + 1, \ldots, 0$ stands for an item set satisfying the structure property of Definition 17. Therefore, we have $p \geq 2^{\kappa-2}T$ if $p > 0$. This is also true for $(p, s, k) \in D^{(k)}$.

Proof. The proof is almost identical to the one of Lemma 21. In fact, the proof is not influenced by only keeping the tuples of smallest size in every profit interval $\tilde{L}^{(\kappa-2)}_\xi$.

Theorem 23. Let $k \in \{0, \ldots, \kappa + 1\}$. For every (non-dominated) entry $(\tilde{p}, \tilde{s}, k) \in F^{(k)}$, there is a tuple $(p, s, k) \in D^{(k)}$ such that

$$p \geq \left( 1 - \frac{\varepsilon}{4 \log_2(\frac{\varepsilon}{4} + 1) + 1} \right)^{\kappa-k+1} \tilde{p} \quad \text{and} \quad s \leq \tilde{s}.$$
Algorithm 3: The approximate dynamic programming

\( D^{(\kappa + 2)} := \{(0, 0, \kappa + 2)\} \);
Backtrack(0, 0, \kappa + 2) := \emptyset;
for \( k = \kappa + 1, \ldots, 0 \) do
\( D^{(k)} := \emptyset; \)
for \((p(\xi), s(\xi), k + 1) \in D^{(k+1)}\) do
    \( (p(\xi), s(\xi), k) := (p(\xi), s(\xi), k + 1); \)
    Backtrack(p(\xi), s(\xi), k) := Backtrack(p(\xi), s(\xi), k + 1);
    \( D^{(k)} := D^{(k)} \cup \{(p(\xi), s(\xi), k)\}; \)
for \( \bar{a} \in \bar{I}^{(k)} \) do
    for \((p, s, k + 1) \in D^{(k+1)} \setminus \{(0, 0, k + 1)\} \) do // Construction of new tuples
        \( (p', s', k) := (p + p(\bar{a}), s + s(\bar{a}), k); \)
        Determine \( \xi' \) for \((p', s', k)\) such that \( p' \in \bar{L}_{\xi'}^{(\kappa - 2)}; \)
        if \( s' < s(\xi') \) or \((p(\xi'), s(\xi'), k) = \emptyset \) then // Only new tuples of smaller size are kept
            \( D^{(k)} := D^{(k)} \setminus \{(p(\xi'), s(\xi'), k)\}; \)
            \( (p(\xi'), s(\xi'), k) := (p', s', k); \)
            Backtrack(p(\xi'), s(\xi'), k) := ((p, s, k + 1), \bar{a});
            \( D^{(k)} := D^{(k)} \cup \{(p(\xi'), s(\xi'), k)\}; \)
if \( k \geq \kappa - 2 \) then // Construction of (possible) tuples \((p(\bar{a}), s(\bar{a}), k)\)
    Determine \( \xi' \) for \( p(\bar{a}) \) such that \( p(\bar{a}) \in \bar{L}_{\xi'}^{(\kappa - 2)}; \)
    if \( s(\bar{a}) < s(\xi') \) or \((p(\xi'), s(\xi'), k) = \emptyset \) then
        \( D^{(k)} := D^{(k)} \setminus \{(p(\xi'), s(\xi'), k)\}; \)
        \( (p(\xi'), s(\xi'), k) := (p(\bar{a}), s(\bar{a}), k); \)
        Backtrack(p(\xi'), s(\xi'), k) := (\bar{a});
        \( D^{(k)} := D^{(k)} \cup \{(p(\xi'), s(\xi'), k)\}; \)
Remove dominated entries from \( D^{(k)}; \)
\[ (p(\xi + 1), s(\xi + 1), k + 1) \]

\[ (p(\xi), s(\xi), k + 1) \]

(a) The tuples in \( D^{(k+1)} \)

\[ (p(\xi) + p(a), s(\xi) + s(a)) \]

\[ +p(a) \]

\[ +s(a) \]

(b) The new tuples are constructed with the items in \( \tilde{I}^{(k)} \).

(c) Only the tuple of smallest size is kept for every \( \tilde{L}^{(k-2)} \).

\[ (p, s) = (p(\xi'), s(\xi'), k) \]

\[ p \]

\[ \lor \]

\[ p_1 \]

\[ p_2 \]

(d) ... and removing the dominated tuples yields \( D^{(k)} \).

Figure 3: The principle of the approximate dynamic programming
There are two possibilities: either $(\bar{p}, \bar{s}, k) \neq (0,0,k)$ because also $(0,0,k) \in D^{(k)}$ (this entry is never removed in the construction of $F^{(k)}$ and $D^{(k)}$).

Suppose now that $(\bar{p}, \bar{s}, k) \neq (0,0,k)$. The theorem is proved by induction for $k = \kappa + 1, \ldots, 0$.

The statement is evident for $k = \kappa + 1$. If $a_{\text{eff-c}}$ exists (i.e. enough copies of $a_{\text{eff}}$ can be glued together without exceeding the capacity $c$), then $F^{(\kappa+1)} = D^{(\kappa+1)} = \{(0,0,\kappa+1), (p(a_{\text{eff-c}}), s(a_{\text{eff-c}}), \kappa+1)\}$. If $a_{\text{eff-c}}$ does not exist, then we even have $F^{(\kappa+1)} = D^{(\kappa+1)} = \{(0,0,\kappa+1)\}$.

Suppose that the statement is true for $k + 1, \ldots, \kappa + 1$. As defined in Lemma 22, $\check{D}^{(k)}$ is the set $D^{(k)}$ before the dominated entries have been removed. Let $(\bar{p}, \bar{s}, k) \in F^{(k)}$.

There are two cases. In the first case, we have $(\bar{p}, \bar{s}, k + 1) \in F^{(k+1)}$. By induction hypothesis, there is a tuple $(p_1, s_1, k + 1) \in D^{(k+1)}$ such that $p_1 \geq \bar{p}(1 - \frac{\varepsilon}{4 \log_2(\frac{\varepsilon}{2}) + 1})^{\kappa-(k+1)+1}$ and $s_1 \leq \bar{s}$ (see Fig. 4(a)). Note that this implies $(p_1, s_1, k + 1) \neq (0,0,k+1)$. Let $\xi_1$ be the index such that $p_1 \in F^{(\kappa-2)}(\kappa_1)$. During the execution of Algorithm 3, $(p_1, s_1, k + 1)$ yields tuple $(p_1, s_1, k)$, which may only be replaced in $\check{D}^{(k)}$ by a tuple of a smaller size, but profit still in $\check{L}^{(\kappa-2)}$. Thus, there must be a tuple $(p_2, s_2, k) \in D^{(k)}$ with $s_2 \leq s_1$ and $p_2 \in \check{L}^{(\kappa-2)}$ (see Fig. 4(b)). Let now $(p, s, k) \in D^{(k)}$ be the tuple that dominates $(p_2, s_2, k)$ (which can of course be $(p_2, s_2, k)$ itself), i.e. $p \geq p_2$ and $s \leq s_2$ (see Fig. 4(c)). For the proof, we have

\[
p \geq p_2 \geq p_1 - 2^{\kappa-2}K \frac{p_1 \neq 0}{p_1} \cdot \left(1 - \frac{2^{\kappa-2}K}{p_1}\right) \geq p_1 \cdot \left(1 - \frac{2^{\kappa-2}K}{2^{\kappa-1}}\right)^{\kappa-k+1}
\]

(1), where $p_1 \cdot \left(1 - \frac{\varepsilon}{4 \log_2(\frac{\varepsilon}{2}) + 1}\right) \geq \bar{p} \cdot \left(1 - \frac{\varepsilon}{4 \log_2(\frac{\varepsilon}{2}) + 1}\right)^{\kappa-k+1}$.

The profit lower bound is therefore true for $(p, s, k)$. As for the size, we have $s \leq s_2 \leq s \leq \bar{s}$ (see also Fig. 4(c)).

Consider now the second case where $(\bar{p}, \bar{s}, k) \in F^{(k)}$, but $(\bar{p}, \bar{s}, k + 1) \notin F^{(k+1)}$. Therefore, $(\bar{p}, \bar{s}, k)$ is a new (non-dominated) tuple such that $(\bar{p}, \bar{s}, k) = (\bar{p} + p(\bar{a}), \bar{s} + s(\bar{a}), k)$ for the right item $\bar{a} \in F^{(k)}$ and tuple $(\bar{p}, \bar{s}, k + 1) \in F^{(k+1)}$. By induction hypothesis, there must be a tuple $(p_1, s_1, k + 1) \in D^{(k+1)}$ such that $p_1 \geq \bar{p}(1 - \frac{\varepsilon}{4 \log_2(\frac{\varepsilon}{2}) + 1})^{\kappa-(k+1)+1}$ and $s_1 \leq \bar{s}$ (see Fig. 5(a)). Thus, the following inequality holds:

\[
p_1 + p(\bar{a}) \geq p(\bar{a}) + \bar{p} \cdot \left(1 - \frac{\varepsilon}{4 \log_2(\frac{\varepsilon}{2}) + 1}\right)^{\kappa-(k+1)+1} \geq \left(1 - \frac{\varepsilon}{4 \log_2(\frac{\varepsilon}{2}) + 1}\right)^{\kappa-(k+1)+1}
\]

There are two possibilities: either $k \geq \kappa - 1$, i.e. $p(\bar{a}) \geq 2^{\kappa-2}T$ holds, and $p_1 + p(\bar{a}) \geq 2^{\kappa-2}T$ directly follows. Otherwise, we have $k \leq \kappa - 3$. Then, $(\bar{p}, \bar{s}, k) = (\bar{p} + p(\bar{a}), \bar{s} + s(\bar{a}), k) \neq (0,0,k)$ implies that $(\bar{p}, \bar{s}, k + 1) \neq (0,0,k+1)$ because the tuple $(0,0,k+1)$ is not used to form any new tuple in $\check{D}^{(k)}$ and therefore in $D^{(k)}$. This again implies that $p_1 \neq 0$ and therefore $p_1 \geq 2^{\kappa-2}T$ as seen in Lemma 22.

Thus, there is an index $\xi_1$ such that $p_1 + p(\bar{a}) \in \check{L}^{(\kappa-2)}$. Similar to above, the tuple $(p_1 + p(\bar{a}), s_1 + s(\bar{a}), k)$ is formed during the construction of $D^{(k)}$ (see Fig. 5(b)). It may only be replaced by a
(a) Since $(\bar{p}, \bar{s}, k + 1) \in F^{(k+1)}$, there must be a tuple $(p_1, s_1, k + 1) \in D^{(k+1)}$ by induction hypothesis whose profit can be bounded from below.

(b) By construction, there must be tuple $(p_2, s_2, k)$ with profit in the same interval $\tilde{l}_{\xi_1}^{(\kappa-2)}$ as $(p_1, s_1, k)$. This allows to bound $p_2$ from below.

(c) There may be a tuple $(p, s, k) \in D^{(k)}$ that dominates $(p_2, s_2, k)$. Since $p \geq p_2$, the bound still holds.

Figure 4: The first case of the proof for Theorem 23 for tuple $(\bar{p}, \bar{s}, k) \in F^{(k)}$, we have $(\bar{p}, \bar{s}, k + 1) \in F^{(k+1)}$. We set $Q := (1 - \frac{\xi}{4 \log_2(\frac{1}{\xi}) + 1})$. 

21
tupel of smaller size. Hence, there must be \((p_2, s_2, k) \in \tilde{D}^{(k)}\) with \(p_2 \in \tilde{L}_{\xi_1}^{(k-2)}\). Let \((p, s, k) \in D^{(k)}\) be the tuple that dominates \((p_2, s_2, k)\) (see Fig. 3(c)). We get

\[
p \geq p_2 \geq p_1 + p(\bar{a}) - 2^{k-2}K \cdot \frac{p_1 + p(\bar{a}) \neq 0}{(p_1 + p(\bar{a}))} \cdot \left(1 - \frac{2^{k-2}K}{p_1 + p(\bar{a})}\right)
\]

\[
p_1 + p(\bar{a}) \geq 2^{k-2}T
\]

\[
\geq \left(\frac{1 - \frac{2^{k-2}K}{2^{k-2}T}}{p_1 + p(\bar{a})}\right) \cdot \left(1 - \frac{\varepsilon}{4 \log_2(\frac{\varepsilon}{\bar{a}}) + 1}\right)
\]

\[
\geq \bar{p} \cdot \left(1 - \frac{\varepsilon}{4 \log_2(\frac{\varepsilon}{\bar{a}}) + 1}\right)^{\kappa - k + 1}.
\]

For the size, we have similar to above \(s \leq s_2 \leq s_1 + s(\bar{a}) \leq \bar{s} + s(\bar{a}) = \bar{s}\) (see also Fig. 3(c)).

**Remark 24.** As can be seen, the proof of Theorem 23 is only possible because it is guaranteed that \(p_1 + p(\bar{a})\) is at least \(2^{k-2}T\). In fact, this is achieved by the construction of the glued item set \(\tilde{I}\) with its structured solution (Theorem 13). Hence, we can prove Lemma 15 and with the introduction of \(a_{\text{eff} - c}\), we have the structure property of Definition 17 with a corresponding solution (Theorem 18). This shows that \(p_1 \geq 2^{k-2}T\) or \(p_1 + p(\bar{a}) \geq 2^{k-2}T\) (see also Lemma 21 and 22). If the tuples did not have this property, a dynamic program like Algorithm 3 would also generate tuples \((p, s, k)\) with \(p < 2^{k-2}T\) for \(k \leq \kappa - 3\). The solution quality could then only be bounded by having a smaller minimum profit than the interval \([\frac{1}{4}P_0, 2P_0]\) and a finer partitioning of it into sub-intervals like \(\tilde{L}_{\xi}^{(k-2)}\). Both would increase the asymptotic running time as can be seen below.

**Corollary 25.** For every \(v \leq c\), there is a tuple \((p, s, 0) \in D^{(0)}\) such that \(s \leq v\) and

\[
p \geq \left(1 - \frac{\varepsilon}{4 \log_2(\frac{\varepsilon}{\bar{a}}) + 1}\right)^{\kappa + 1} \cdot OPT_{\text{St}}(\tilde{I} \cup \{a_{\text{eff} - c}\}, v).
\]

**Proof.** Lemma 21 states that there is a \((\bar{p}, \bar{s}, 0) \in F^{(0)}\) such that \(\bar{p} = OPT_{\text{St}}(\tilde{I} \cup \{a_{\text{eff} - c}\}, v)\) and \(\bar{s} \leq v\). Theorem 23 then implies that there is a tuple \((p, s, 0) \in D^{(0)}\) with the desired property.

**Theorem 26.** Algorithm 5 constructs all tuple sets \(D^{(k)}\) for \(k = \kappa + 1, \ldots, 0\) in time \(O\left(\frac{1}{\bar{a}} \log^3 \frac{1}{\varepsilon}\right)\).

The space needed for the algorithm and to save the \(D^{(k)}\) as well the backtracking information is in \(O\left(\frac{1}{\bar{a}} \log^2 \frac{1}{\varepsilon}\right)\).

**Proof.** The profit interval \([\frac{1}{4}P_0, 2P_0]\) is partitioned into \(O(\xi_0)\) many intervals \(\tilde{L}_{\xi}^{(k-2)}\). \(D^{(k)}\) saves at most one tuple with the corresponding backtracking information for every \(\tilde{L}_{\xi}^{(k-2)}\) or the information that a tuple does not exist. Thus, the space needed for all \(D^{(k)}\) and the corresponding backtracking data is in \(O(\kappa \cdot \xi_0) = O(\kappa \cdot (\xi_{2\kappa})) = O(\log(\frac{1}{\varepsilon}) \cdot (\log(\frac{1}{\varepsilon}) \frac{1}{2})) = O(\frac{1}{\varepsilon} \log^2 \frac{1}{\varepsilon})\). All other information of the algorithm is only temporarily saved and needs \(O(1)\).

The loops dominate the running time. Apart from removing the dominated entries, they need in total \(O(\kappa \cdot (\xi_0 + \xi_0 \cdot |\tilde{I}^{(k)}| + |\tilde{I}^{(k)}|))\) which is \(O(\log(\frac{1}{\varepsilon}) (\frac{1}{2} \log(\frac{1}{\varepsilon}) \frac{1}{2} \log(\frac{1}{\varepsilon}))) = O\left(\frac{1}{\varepsilon} \log^3 \frac{1}{\varepsilon}\right)\). As stated in 20 and 11 Lemma 5], non-dominated tuples \((p, s, k)\) can be removed in linear time in the number of tuples if the entries are different and sorted by profit. This is the case because every tuple in \(D^{(k)}\) is stored in an array sorted according to the corresponding \(\xi\). The total time for removing the dominated entries from all \(D^{(k)}\) is therefore in \(O(\kappa \cdot \xi_0) = O(\frac{1}{\varepsilon} \log^2 \frac{1}{\varepsilon})\), which is dominated by the overall running time.
(a) Since $(\bar{p}, \bar{s}, k + 1) \notin F^{(k+1)}$, there must be an item $\tilde{a}$ such that $(\bar{p}, \bar{s}, k) = (\tilde{p} + p(\tilde{a}), \tilde{s} + s(\tilde{a}), k)$ for $(\tilde{p}, \tilde{s}, k + 1) \in F^{(k+1)}$. By induction hypothesis, there must be a tuple $(p_1, s_1, k + 1) \in D^{(k+1)}$ whose profit can be bounded from below.

(b) The tuple $(p_1 + p(\tilde{a}), s_1 + s(\tilde{a}), k)$ is constructed during the execution of the dynamic program.

(c) Like in the first case, there must be a tuple $(p, s, k) \in D^{(k)}$ whose profit can be bounded as desired. Here, $(p_2, s_2, k)$ is not dominated, i.e. $(p, s, k) = (p_2, s_2, k)$.

Figure 5: The second case of the proof for Theorem 23: we have $(\bar{p}, \bar{s}, k + 1) \notin F^{(k+1)}$, but $(\bar{p}, \bar{s}, k) \in F^{(k)}$. We set $Q := (1 - \frac{\varepsilon}{4 \log_4(\frac{1}{\varepsilon}) + 1})$. 

23
6 The Algorithm

We can now put together the entire approximation algorithm.

**Algorithm 4:** The complete algorithm

**Input:** Item set $I$

**Output:** Profit $P$, solution set $J$

Determine $P_0$ and define $T, K$;

Partition the items into $I_L$ and $I_S$ and find $a_{\text{eff}}$;

if Item $a$ found during partition with $p(a) = 2P_0$ then

return $2P_0, \{a\}$;

Reduce $I_L$ to $I_L, \text{red}$ with Algorithm 1;

Construct $\tilde{I}$ with Algorithm 2 and the item $a_{\text{eff}} - c$;

if $p(\tilde{a}_0^\kappa) = P_0$ and $s(\tilde{a}_0^\kappa) \leq \frac{c}{2}$ then

Recursively undo the gluing of $\tilde{a}_0^\kappa$ to get item set $J'$.
Let $J$ be the set consisting of two copies of every item in $J'$;

return $2P_0, J$;

Construct with Algorithm 3 the tuple sets $D^{(\kappa+1)}, \ldots, D^{(0)}$;

Find $(p, s, 0) \in D^{(0)}$ such that

$P := p + \text{OPT}(\{a_{\text{eff}}\}, c - s) = \max_{(p', s', 0) \in D^{(0)}} p' + \text{OPT}(\{a_{\text{eff}}\}, c - s')$;

Backtrack the tuple $(p, s, 0)$ to find the corresponding structured set $J' \subseteq \tilde{I}$;

Recursively undo the gluing of all $\tilde{a} \in J'$ and add these items to solution set $J$;

Add the items of $\text{OPT}(\{a_{\text{eff}}\}, c - s)$ to $J$;

return $P, J$;

**Theorem 27.** Algorithm 4 finds a solution of value at least $(1 - \varepsilon)\text{OPT}(I)$.

**Proof.** The algorithm returns a feasible solution: $(p, s, 0)$ represents an item set of size $s$. If items $\tilde{a} \in \tilde{I}$ derived from gluing are part of the solution, their ungluing does not change the total size nor the total profit (see Remark 11).

We prove the solution quality. First, the algorithm considers the two special cases listed at the beginning of Section 5. In the third case, let $v$ be the volume from Theorem 18 Corollary 25 guarantees the existence of one $(p, s, 0) \in D^{(0)}$ with $s \leq v$ such that

$$p \geq \left(1 - \varepsilon, \frac{1}{4 \log_2 \left(\frac{2}{\varepsilon}\right)} + 1\right)^{\kappa+1} \text{OPT}_{\text{St}} \left(\tilde{I} \cup \{a_{\text{eff}} - c\}, v\right).$$

Moreover, we have $\text{OPT}(\{a_{\text{eff}}\}, c - s) \geq \text{OPT}(\{a_{\text{eff}}\}, c - v)$ because $c - s \geq c - v$. Thus, the
following inequality holds for this \((p,s,0)\):

\[
p + OPT(\{a_{\text{eff}}\}, c - s) \geq \left(1 - \frac{\varepsilon}{4 \log_2(\frac{\varepsilon}{2}) + 1}\right)^{\kappa + 1} OPT_{St}(\tilde{I} \cup \{a_{\text{eff}} - c\}, v) + OPT(\{a_{\text{eff}}\}, c - v)
\]

\[
\geq \left(1 - \frac{\varepsilon}{4 \log_2(\frac{\varepsilon}{2}) + 1}\right)^{\kappa + 2} OPT(I) - \left(\left(1 - \frac{\varepsilon}{4 \log_2(\frac{\varepsilon}{2}) + 1}\right)^{\kappa + 1} T
\]

\[
\geq \left(1 - \frac{\varepsilon}{4 \log_2(\frac{\varepsilon}{2}) + 1}\right)^{2\kappa + 2} OPT(I) - \frac{1}{2} \varepsilon P_0
\]

\[
\geq \left(1 - \frac{\varepsilon}{4 \log_2(\frac{\varepsilon}{2}) + 1}\right)^{\kappa + 1} OPT(I) - \frac{1}{2} \varepsilon OPT(I)
\]

\[
= (1 - \varepsilon) OPT(I)
\]

Taking the maximum over all \((p,s,0) \in D(0)\) therefore yields the desired solution. \(\square\)

**Remark 28.** The total bound of the approximation ratio is mainly due to the exponent \(2\kappa + 2\), i.e. that we make the multiplicative error of \((1 - \frac{\varepsilon}{4 \log_2(\frac{\varepsilon}{2}) + 1})\) only \(2\kappa + 2\) times. Such an error occurs when \(I\) is replaced by \(I_{L,\text{red}}\) at the beginning (Lemma 7), in each of the \(\kappa\) iterations in which \(\tilde{I}\) is constructed (Theorem 13), and in \(\kappa + 1\) of the \(\kappa + 2\) iterations of the dynamic program (Theorem 23 and Corollary 25). The error of the dynamic program can be bounded because the structured solution has at least one item of profit at least \(2\kappa - 2\) \(T\) (see the second property of Definition 17 and Remark 24).

**Theorem 29.** The algorithm has a running time in \(O(n + \frac{1}{\varepsilon} \log^3 \frac{1}{\varepsilon})\) and needs space in \(O(n + \frac{1}{\varepsilon} \log^2 \frac{1}{\varepsilon})\).

**Proof.** Determining \(P_0\), constructing \(I_L\) and \(I_S\) as well as finding \(a_{\text{eff}}\) can all be done in time and space \(O(n)\) as stated in Theorems 4 and 5. The definition of \(T\) and \(K\) in time and space \(O(1)\) is obvious. It is also clear that an item \(p(a) = 2P_0\) can directly be found during the construction of \(I_L\) such that the \(s\) if-condition does not influence the asymptotic running time.

The set \(I_{L,\text{red}}\) is returned by Algorithm 1 in time \(O(n + \frac{1}{\varepsilon} \log^2 \frac{1}{\varepsilon})\) and space \(O(\frac{1}{\varepsilon} \log^2 \frac{1}{\varepsilon})\) (see Theorem 8).

Algorithm 2 constructs the \(\tilde{I}^{(k)}\) and \(\tilde{I}\) in time \(O(\frac{1}{\varepsilon} \log^3 \frac{1}{\varepsilon})\) and space \(O(\frac{1}{\varepsilon} \log^2 \frac{1}{\varepsilon})\) as explained by Theorem 19 which clearly dominates the construction of \(a_{\text{eff}} - c\) in \(O(1)\).

The second if-condition can be checked in \(O(1)\). The running time for undoing the gluing will be determined at the end of the proof.

Algorithm 3 constructs the sets \(D^{(k)}\) in time \(O(\frac{1}{\varepsilon} \log^3 \frac{1}{\varepsilon})\) and space \(O(\frac{1}{\varepsilon} \log^2 \frac{1}{\varepsilon})\) (see Theorem 26). For one \((p', s', 0)\), the corresponding \(OPT(\{a_{\text{eff}}\}, c - s')\) can be found in \(O(1)\) by \(\left\lfloor \frac{c - s'}{s(a_{\text{eff}})} \right\rfloor \cdot p(a_{\text{eff}})\).
Thus, finding the best \((p, s, 0)\) can be done in \(O(|D^{(0)}|) = O(\epsilon) = O(\frac{1}{\epsilon} \log \frac{1}{\epsilon})\). Since only the currently best \((p, s, 0)\) has to be saved, the space needed is in \(O(1)\).

The backtracking for \((p, s, 0)\) is in \(O(\kappa) = O(\log \frac{1}{\epsilon})\): every entry Backtrack\((p', s', k)\) for \(k = 0, \ldots, \kappa + 1\) states whether the tuple was formed by adding an item \(\bar{a} \in \bar{I}^{(k)}\) and with which tuple \((p'', s'', k + 1)\) to continue. Hence, the item set \(J'\) also has at most \(O(\log \frac{1}{\epsilon})\) items in \(\bar{I} \cup \{a_{\text{eff}} - c\}\), which bounds the storage space needed.

To conclude, the time and space for the ungluing still have to be bounded. Consider one item \(\bar{a} \in \bar{I}\). The backtracking information Backtrack\((\bar{a})\) returns two items \((\bar{a}_1, \bar{a}_2)\) (with \(\bar{a}_1, \bar{a}_2 \in I_{L, \text{red}} \cup \bar{I}\)) on which the backtracking can be recursively applied. The recursive ungluing of the items can be represented as a binary tree where the root is the original item \(\bar{a}\) and the (two) children of each node are the items \((\bar{a}', \bar{a}'')\) returned by the backtracking information. The leaves of the tree are the original items in \(I_{L, \text{red}}\). This binary tree obviously has height in \(O(\kappa)\) because the children \((\bar{a}', \bar{a}'')\) for one \(\bar{a} \in \bar{I}^{(k)}\) are in \(\bar{I}^{(k-1)} \cup I_{L, \text{red}}\). A binary tree of height \(O(\kappa) = O(\log \frac{1}{\epsilon})\) has at most \(O(\frac{1}{\epsilon})\) nodes. The backtracking or ungluing of \(\bar{a}\) therefore can be done in time and space \(O(\frac{1}{\epsilon})\), which also includes saving the items \(\bar{a} \in I_{L, \text{red}}\) of which \(\bar{a}\) is built. Since \(J'\) has \(O(\log \frac{1}{\epsilon})\) items, the original items \(I_{L, \text{red}}\) of the approximate solution can be found in time and space \(O(\frac{1}{\epsilon} \log \frac{1}{\epsilon})\). This also dominates the time to undo the gluing of \(a_0^{(\kappa)}\) should the body of the second if-condition be executed.

Similar to above, the number of items \(OPT\{\{a_{\text{eff}}\}, c - s\}\) can be found in \(O(1)\). To sum up, Algorithm 4 has the running time and a space complexity as stated.

### 7 Concluding Remarks

The most important steps in this algorithm are the creation of the item set \(\bar{I}\) by gluing and the introduction of \(a_{\text{eff}} - c\). This guarantees the existence of an approximate solution with the structure of Definition 17. Therefore, the approximate dynamic programming has to store less tuples \((p, s, k)\) than in the case without the structure.

We have already extended our algorithm to the Unbounded Knapsack Profit with Inversely Proportional Profits (UKPIP) introduced in [11]. Here, several knapsack sizes \(0 < c_1 < \ldots < c_M = 1\) are given, and the profit of an item counts as \(p_i / c_i\) if packed in \(c_i\). The goal is to find the best knapsack size and corresponding solution of maximum profit. UKPIP is used for column generation in our AFPTAS for Variable-Sized Bin Packing [10] where several bin sizes are given and the goal is to minimize the total volume of the bins used. The faster FPTAS for UKPIP yields a faster AFPTAS for Variable-Sized Bin Packing [12].

There are interesting open questions. As stated in Subsection 1.2, the space complexity is a more serious bottleneck than the running time. Recently, Lokshtanov and Nederlof [21] have shown that the Subset Sum Problem has a pseudo-polynomial time and only polynomial space algorithm. Subset Sum is a special case of Knapsack where the profit of an item is equal to its size, i.e. \(p_j = s_j\). Moreover, it was shown that Unary Subset Sum is in Logspace [3, 13]. Gál et al. [4] describe an FPTAS for Subset Sum whose space complexity is in \(O(\frac{1}{\epsilon})\), i.e. which does not depend on the actual input size, and whose running time is in \(O(\frac{1}{\epsilon} n (n + \log n + \log \frac{1}{\epsilon}))\). Can any of these results be further extended to improve the space complexity of an UKP FPTAS? Finally, it is open whether the ideas presented in this paper can be extended to the normal 0-1 KP or other KP variants as well as used for column generation of other optimization problems. The currently fastest known
algorithm for 0-1 KP is due to Kellerer and Pferschy \cite{16,18}. We mention in closing that by using the same approach similar improved approximation algorithms can be expected for various Packing and Scheduling Problems, e.g. for Bin Covering, Bin Packing with Cardinality Constraints, Scheduling Multiprocessor Tasks and Resource-constrained Scheduling.

References

[1] R. E. Bellman. Dynamic Programming. Princeton University Press, 1957.

[2] M. Bougeret, P.-F. Dutot, K. Jansen, C. Robenek, and D. Trystram. “Approximation Algorithms for Multiple Strip Packing and Scheduling Parallel Jobs in Platforms”. In: Discrete Math., Alg. and Appl. 3.4 (2011), pp. 553–586.

[3] M. Elberfeld, A. Jakoby, and T. Tantau. Logspace Versions of the Theorems of Bodlaender and Courcelle. Tech. rep. 62. Electronic Colloquium on Computational Complexity (ECCC), 2010. First published in 51th Annual Symposium on Foundations of Computer Science (FOCS 2010). IEEE Computer Society, 2010, pp. 143–152.

[4] A. Gál, J. Jang, N. Limaye, M. Mahajan, and K. Sreenivasasiah. Space-Efficient Approximations for Subset Sum. Tech. rep. 180. Electronic Colloquium on Computational Complexity (ECCC), 2014.

[5] M. Garey and D. Johnson. Computers and Intractability. A Guide to the Theory of NP-Completeness. W.H. Freeman and Company, 1979.

[6] P. C. Gilmore and R. E. Gomory. “A Linear Programming Approach to the Cutting Stock Problem”. In: Oper. Res. 9.6 (1961), pp. 849–859.

[7] M. D. Grigoriadis, L. G. Khachiyan, L. Porkolab, and J. Villavicencio. “Approximate Max-Min Resource Sharing for Structured Concave Optimization”. In: SIAM J. Optim. 11.4 (2001), pp. 1081–1091.

[8] O. H. Ibarra and C. E. Kim. “Fast approximation algorithms for the knapsack and sum of subset problems”. In: J. ACM 22 (1975), pp. 463–468.

[9] K. Jansen. “Approximation Algorithms for Min-Max and Max-Min Resource Sharing Problems, and Applications”. In: Efficient approximation and online algorithms. Vol. 3484. LNCS. Springer, 2006, pp. 156–202.

[10] K. Jansen and S. Kraft. “An Improved Approximation Scheme for Variable-Sized Bin Packing”. In: Proceedings of the 37th International Symposium on Mathematical Foundations of Computer Science, MFCS 2012. Vol. 7464. LNCS. Springer, 2012, pp. 529–541.

[11] K. Jansen and S. Kraft. “An Improved Knapsack Solver for Column Generation”. In: Proceedings of the 8th International Computer Science Symposium in Russia, CSR 2013. Vol. 7913. LNCS. Springer, 2013, pp. 12–23.

[12] K. Jansen and S. E. J. Kraft. “A Faster FPTAS for the Unbounded Knapsack Problem with Inversely Proportional Profits”. 2015. The result will be published in the PhD thesis of S. E. J. Kraft.

[13] D. M. Kane. Unary Subset-Sum is in Logspace. 2010. arXiv:1012.1336
[14] N. Karmarkar and R. M. Karp. “An Efficient Approximation Scheme for the One-Dimensional Bin-Packing Problem”. In: 23rd Annual Symposium on Foundations of Computer Science (FOCS 1982). IEEE Computer Society, 1982, pp. 312–320.

[15] H. Kellerer, R. Mansini, U. Pferschy, and M. G. Speranza. “An efficient fully polynomial approximation scheme for the Subset-Sum Problem”. In: J. Comput. Syst. Sci. 66.2 (2003), pp. 349–370.

[16] H. Kellerer and U. Pferschy. “A New Fully Polynomial Time Approximation Scheme for the Knapsack Problem”. In: J. Comb. Optim. 3.1 (1999), pp. 59–71.

[17] H. Kellerer and U. Pferschy. “Improved Dynamic Programming in Connection with an FTPAS for the Knapsack Problem”. In: J. Comb. Optim. 8.1 (2004), pp. 5–11.

[18] H. Kellerer, U. Pferschy, and D. Pisinger. Knapsack Problems. Springer, 2004.

[19] C. Kenyon and E. Rémila. “A Near-Optimal Solution to a Two-Dimensional Cutting Stock Problem”. In: Mathematics of Operations Research 25.4 (2000), pp. 645–656. First published as “Approximate Strip Packing”. In: 37th Annual Symposium on Foundations of Computer Science (FOCS 1996). IEEE Computer Society, 1996, pp. 31–36.

[20] E. L. Lawler. “Fast Approximation Algorithms for Knapsack Problems”. In: Math. Oper. Res. 4.4 (1979), pp. 339–356.

[21] D. Lokshtanov and J. Nederlof. “Saving space by algebraization”. In: Proceedings of the 42nd ACM Symposium on Theory of Computing, STOC 2010. ACM, 2010, pp. 321–330.

[22] S. A. Plotkin, D. B. Shmoys, and É. Tardos. “Fast Approximation Algorithms for Fractional Packing and Covering Problems”. In: Math. Oper. Res. 20 (1995), pp. 257–301. First published in 32nd Annual Symposium on Foundations of Computer Science (FOCS 1991). IEEE Computer Society, 1991, pp. 495–504.

[23] H. Shachnai and O. Yehezkely. “Fast Asymptotic FPTAS for Packing Fragmentable Items with Costs”. In: Proceedings of the 16th International Symposium on Fundamentals of Computation Theory, FCT 2007. Vol. 4639. LNCS. Springer, 2007, pp. 482–493.

[24] M. Sviridenko. “A note on the Kenyon-Remila strip-packing algorithm”. In: Inf. Process. Lett. 112.1–2 (2012), pp. 10–12.

This bibliography contains information from the DBLP database (www.dblp.org), which is licensed under the ODC Attribution License.