

Exact Counting and Sampling of Optima for the Knapsack Problem

Jakob Bossek¹[0000–0002–4121–4668], Aneta Neumann²[0000–0002–0036–4782], and Frank Neumann²[0000–0002–2721–3618]

¹ Statistics and Optimization, University of Münster, Germany

² School of Computer Science, The University of Adelaide, Australia
bossek@wi.uni-muenster.de, {aneta, frank}.neumann@adelaide.edu.au

Abstract. Computing sets of high quality solutions has gained increasing interest in recent years. In this paper, we investigate how to obtain sets of optimal solutions for the classical knapsack problem. We present an algorithm to count exactly the number of optima to a zero-one knapsack problem instance. In addition, we show how to efficiently sample uniformly at random from the set of all global optima. In our experimental study, we investigate how the number of optima develops for classical random benchmark instances dependent on their generator parameters. We find that the number of global optima can increase exponentially for practically relevant classes of instances with correlated weights and profits which poses a justification for the considered exact counting problem.

Keywords: Zero-one knapsack problem · exact counting · sampling · dynamic programming

1 Introduction

Classical optimisation problems ask for a single solution that maximises or minimises a given objective function under a given set of constraints. This scenario has been widely studied in the literature and a vast amount of algorithms are available. In the case of NP-hard optimisation problems one is often interested in a good approximation of an optimal solution. Again, the focus here is on a single solution.

Producing a large set of optimal (or high quality) solutions allows a decision maker to pick from structurally different solutions. Such structural differences are not known when computing a single solution. Computing a set of optimal solutions has the advantage that more knowledge on the structure of optimal solutions is obtained and that the best alternative can be picked for implementation. As the number of optimal solutions might be large for a given problem, sampling from the set of optimal solutions provides a way of presenting different alternatives.

Related to the task of computing the set of optimal solutions, is the task of computing diverse sets of solutions for optimisation problems. This area of

research has obtained increasing attention in the area of planning where the goal is to produce structurally different high quality plans [19,11,10,9]. Furthermore, different evolutionary diversity optimisation approaches which compute diverse sets of high quality solutions have been introduced [20,14,15]. For the classical traveling salesperson problem, such an approach evolves a diverse set of tours which are all a good approximation of an optimal solution [3].

Counting problems are frequently studied in the area of theoretical computer science and artificial intelligence [2,8,6,7]. Here the classical goal is to count the number of solutions that fulfill a given property. This might include counting the number of optimal solutions. Many counting problems are $\#P$ -complete [21] and often approximations of the number of such solutions, especially approximations on the number of optimal solutions are sought [5]. Further examples include counting the number of shortest paths between two nodes in graphs [13] or exact counting of minimum-spanning-trees [1]. For the knapsack problem (KP), the problem of counting the number of feasible solutions, i.e. the number of solutions that do not violate the capacity constraint, is $\#P$ -complete. As a consequence, different counting approaches have been introduced to approximately count the number of feasible solutions [4,18,22].

In this paper, we study the classical zero-one knapsack problem (KP). We develop an algorithm that is able to compute all optimal solutions for a given knapsack instance. The algorithm adapts the classical dynamic programming approach for KP in the way that all optimal solutions are produced implicitly. As the number of such solutions might grow exponentially with the problem size for some instances, we develop a sampling approach which samples solutions for the set of optimal solutions uniformly at random.

We carry out experimental investigations for different classes of knapsack instances given in the literature (instances with uniform random weights and instances with different correlation between weights and profits). Using our approach, we show that the number of optimal solutions significantly differs between different knapsack instance classes. In particular, for instances with correlated weights and profits – a group of great importance in practical applications – an exponential growth of optima is observed. In addition, we point out that changing the knapsack capacity slightly can reduce the number of optimal solutions from exponential to just a single solution.

The paper is structured as follows. In the next section, we introduce the task of computing the set of optimal solutions for the knapsack problem. Afterwards, we present the dynamic programming approach for computing the set of optimal solutions and show how to sample efficiently from the set of optimal solutions without having to construct the whole set of optimal solutions. In our experimental investigations, we show the applicability of our approach to a wide range of knapsack instances and point out insights regarding the number of optimal solutions for these instances. Finally, we finish with some concluding remarks.

2 Problem Formulation

We now introduce the problem of computing all optimal solutions for the knapsack problem. In the following, we use the standard notation $[n] = \{1, \dots, n\}$ to express the set of the first n positive integers. The problem studied is the classical NP-hard *zero-one knapsack problem* (KP). We are given a knapsack with integer capacity $W > 0$ and a finite set of n items, each with positive integer weight w_i and associated integer profit or value v_i for $i \in [n]$. Each subset $s \subset [n]$ is called a *solution/packing*. We write

$$w(s) = \sum_{i \in s} w_i \text{ and } v(s) = \sum_{i \in s} v_i$$

for the total weight and value respectively. A solution is *feasible* if its total weight does not exceed the capacity. Let

$$\mathcal{S} = \{s \mid s \subset [n] \wedge w(s) \leq W\}$$

be the set of feasible solutions. The goal in the optimisation version of the problem is to find a solution $s^* \in \mathcal{S}$ such that

$$s^* = \arg \max_{s \in \mathcal{S}} v(s).$$

Informally, in the optimisation version of the KP, we strive for a subset of items that maximises the total profit under the constraint that the total weight remains under the given knapsack capacity.

Let v_{\max} be the value of an optimal solution s^* and let

$$\mathcal{S}^* = \{s \mid s \in \mathcal{S} \wedge v(s) = v_{\max}\}$$

be the set of optimal solutions. In this work we study a specific counting problem which we refer to as $\#KNAPSACK^*$ in the following. Here, the goal is to determine *exactly* the cardinality of the set \mathcal{S}^* . Note that this is a special case of the classic counting version $\#KNAPSACK$ where we aim to count the set of all feasible solutions \mathcal{S} , and $\mathcal{S}^* \subset \mathcal{S}$. In addition, we are interested in procedures to sample uniformly at random a subset of k out of $|\mathcal{S}^*|$ solutions with $k \leq |\mathcal{S}^*|$.

3 Exact Counting and Sampling of Optima

In this section we introduce the algorithms for the counting and sampling problems stated. We first recap the classic dynamic programming algorithm for the zero-one KP as it forms the foundation for our algorithm(s).

3.1 Recap: Dynamic Programming for the KP

Our algorithms are based on the dynamic programming approach for the optimisation version (see, e.g. the book by Kellerer et al. [12]). This well-known algorithm maintains a table V with components $V(i, w)$ for $0 \leq i \leq n, 0 \leq w \leq W$.

Here, component $V(i, w)$ holds the maximum profit that can be achieved with items up to item i , i.e., $\{1, \dots, i\}$, and capacity w . The table is constructed bottom-up following the recurrence

$$V(i, w) = \max \left\{ \underbrace{V(i-1, w)}_{\text{(a) leave item } i}, \underbrace{V(i-1, w - w_i) + v_i}_{\text{(b) take item } i} \right\}$$

for $1 \leq i \leq n, 0 \leq w \leq W$. Essentially, the optimal value $V(i, w)$ is achieved by making a binary decision for every item $i \in [n]$ relying on pre-calculated optimal solutions to sub-problems with items from $\{1, \dots, i-1\}$. The options are (a) either leaving item i where the optimal solution is realised by the maximum profit achieved with items $\{1, \dots, i-1\}$ and capacity w . Option (b) deals with putting item i into the knapsack (only possible if $w_i < w$) gaining profit v_i at the cost of additional w_i units of weight. In consequence, the optimal profit $V(i, w)$ is the optimal profit with items from $\{1, \dots, i-1\}$ and capacity $w - w_i$, i.e., $V(i, w) = V(i-1, w - w_i) + v_i$. Initialization follows

$$V(0, w) = 0 \quad \forall 0 \leq w \leq W \quad (1)$$

$$V(i, w) = -\infty \quad \forall w < 0 \quad (2)$$

which covers the base cases of an empty knapsack (Eq. (1)) and a negative capacity, i.e., invalid solution (Eq. (2)), respectively. Eventually, $V(n, W)$ holds the profit of an optimal solution.

3.2 Dynamic Programming for #KNAPSACK*

We observe that if $V(i-1, w) = V(i-1, w - w_i) + v_i$ we can achieve *the same maximum profit* $V(i, w)$ by both options (a) or (b). Analogously to $V(i, w)$ let $C(i, w)$ be the number of solutions with maximum profit given items from $\{1, \dots, i\}$ and capacity w . Then there are three update options for $C(i, w)$ for $1 \leq i \leq n$ and $0 \leq w \leq W$ (we discuss the base case later): (a') Either, as stated above, we can obtain the same maximum profit $V(i, w)$ by packing or not packing item i . In this case, $C(i, w) = C(i-1, w) + C(i-1, w - w_i)$ since the item sets leading to $V(i-1, w)$ and $V(i-1, w - w_i)$ are necessarily disjoint by construction. Options (b') and (c') correspond to (a) and (b) leading to the recurrence

$$C(i, w) = \begin{cases} C(i-1, w) + C(i-1, w - w_i) & \text{if } V(i-1, w) = V(i-1, w - w_i) + v_i \\ C(i-1, w) & \text{if } V(i-1, w) > V(i-1, w - w_i) + v_i \\ C(i-1, w - w_i) & \text{otherwise.} \end{cases} \quad (3)$$

Analogously to Equations (1) and (2) the bases cases

$$C(0, w) = 1 \quad \forall 0 \leq w \leq W \quad (4)$$

$$C(i, w) = 0 \quad \forall w < 0 \quad (5)$$

Algorithm 1: DP-algorithm for #KNAPSACK*

Input: Number of items n , capacity W

```

1 for  $w \leftarrow 0$  to  $W$  do
2    $V(0, w) \leftarrow 0$ ;
3    $C(0, w) \leftarrow 1$ ;
4 for  $i \leftarrow 0$  to  $n$  do
5    $V(i, 0) \leftarrow 0$ ;
6    $C(i, 0) \leftarrow 1$ ;
7 for  $i \leftarrow 1$  to  $n$  do
8   for  $w \leftarrow 1$  to  $W$  do
9     if  $w_i > w$  then
10       $V(i, w) \leftarrow V(i-1, w)$ ;
11       $C(i, w) \leftarrow C(i-1, w)$ ;
12     else
13       if  $V(i-1, w) = V(i-1, w-w_i) + v_i$  then
14          $V(i, w) \leftarrow V(i-1, w)$ ;
15          $C(i, w) \leftarrow C(i-1, w) + C(i-1, w-w_i)$ ;
16       else if  $V(i-1, w) > V(i-1, w-w_i) + v_i$  then
17          $V(i, w) \leftarrow V(i-1, w)$ ;
18          $C(i, w) \leftarrow C(i-1, w)$ ;
19       else
20          $V(i, w) \leftarrow V(i-1, w-w_i) + v_i$ ;
21          $C(i, w) \leftarrow C(i-1, w-w_i)$ ;
22 return  $V, C$ 

```

handle the empty knapsack (the empty-set is a valid solution) and the case of illegal items (no valid solution(s) at all). The base cases are trivially correct. The correctness of the recurrence follows inductively by the preceding argumentation. Hence, we wrap up the insights in the following lemma whose proof is embodied in the preceding paragraph.

Lemma 1. *Following the recurrence in Eq. (3) $C(i, w)$ stores the number of optimal solutions using items from $\{1, \dots, i\}$ given the capacity w .*

Algorithm 1 shows pseudo-code for our algorithm. The algorithm directly translates the discussed recurrences for tables V and C . Note that the pseudo-code is not optimised for performance and elegance, but for readability.

Theorem 1. *Let \mathcal{S}^* be the set of optimal solutions to a zero-one knapsack problem with n items and capacity W . There is a deterministic algorithm that calculates $|\mathcal{S}^*|$ exactly with time- and space-complexity $O(n^2W)$.*

Proof. The correctness follows from Lemma 1. For the space- and time-complexity note that two tables with dimensions $(n+1) \times (W+1)$ are filled. For Table V the algorithm requires constant time per cell and hence time and space $O(nW)$.

Table 1. Exemplary knapsack instance (left) and the dynamic programming tables $V(i, w)$ (center) and $C(i, w)$ (right) respectively. Table cells highlighted in light-gray indicate components where two options are possible: either packing item i or not. Light-green cells indicate the total value of any optimal solution for $V(\cdot, \cdot)$ and $|\mathcal{S}^*|$ for $C(\cdot, \cdot)$.

$W = 8$			$V(\cdot, \cdot)$										$C(\cdot, \cdot)$									
			$i \backslash w$	0	1	2	3	4	5	6	7	8	$i \backslash w$	0	1	2	3	4	5	6	7	8
i	w_i	v_i	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1
1	3	3	1	0	0	0	3	3	3	3	3	3	1	1	1	1	1	1	1	1	1	1
2	8	10	2	0	0	0	3	3	3	3	3	10	2	1	1	1	1	1	1	1	1	1
3	2	3	3	0	0	3	3	3	6	6	6	10	3	1	1	1	2	2	1	1	1	1
4	2	4	4	0	0	4	4	7	7	7	10	10	4	1	1	1	1	1	2	2	1	2
5	2	3	5	0	0	4	4	7	7	10	10	10	5	1	1	1	1	2	3	1	3	4

Table C stores the number of solutions which can be exponential in the input size n as we shall see later. Therefore, $C(i, j) \leq 2^n$ and $O(\log(2^n)) = O(n)$ bits are necessary to encode these numbers. Thus, the addition in line 15 in Algorithm 1 requires time $O(n)$ which results in space and time requirement of $O(n^2W)$. \square

Note that the complexity reduces to $O(nW)$ if $|\mathcal{S}^*| = \text{poly}(n)$. Furthermore, the calculation of row i only relies on values in row $i - 1$. Hence, we can reduce the complexity by a factor of n if only two rows of V and C are stored and the only value of interest is the number of optimal solutions.

For illustration we consider a simple knapsack instance with $n = 5$ items fully described by the left-most table in Table 1. Let $W = 8$. In this setting there exist four optima $s_1 = \{2\}$, $s_2 = \{1, 4, 5\}$, $s_3 = \{1, 3, 4\}$ and $s_4 = \{3, 4, 5\}$ with profit 10 each and thus $|\mathcal{S}^*| = 4$. The dynamic programming tables are shown in Table 1 (center and right). For improved visual accessibility we highlight table cells where both options (packing item i or not) are applicable and hence an addition of the number of combinations of sub-problems is performed in C by the algorithm (cf. first case in the recurrence in Eq. 3).

3.3 Uniform Sampling of Optimal Solutions

The DP algorithm introduced before allows to count $|\mathcal{S}^*|$ and hence to solve $\#\text{KNAPSACK}^*$ exactly. The next logical step is to think about a sampler, i.e., an algorithm that samples uniformly at random from \mathcal{S}^* even if \mathcal{S}^* is exponential in size. In fact, we can utilize the tables C and V for this purpose as they implicitly encode the information on all optima. A similar approach was used by Dyer [4] to approximately sample from the (approximate) set of feasible solutions in his dynamic programming approach for $\#\text{KNAPSACK}$. Our sampling algorithm however is slightly more involved due to a necessary case distinctions.

Algorithm 2: Uniform Sampling of Optima

Input: DP tables V and C (see Algorithm 1), number of items n , capacity W , desired number of solutions k .

```

1  $S \leftarrow \emptyset$ ;
2 while  $k > 0$  do
3    $L \leftarrow \emptyset$ ;
4    $i \leftarrow n$ ;
5    $w \leftarrow W$ ;
6   while  $i > 0$  and  $w > 0$  do
7     if  $w_i \leq w \wedge V(i, w) = V(i-1, w) \wedge V(i, w) = V(i-1, w-w_i) + v_i$ 
8       then
9          $q \leftarrow \frac{C(i-1, w-w_i)}{C(i, w)}$ ;
10        Let  $r$  be a random number in  $[0, 1]$ ;
11        if  $r < q$  then
12           $L \leftarrow L \cup \{i\}$ ;
13           $w \leftarrow w - w_i$ ;
14        else if  $V(i, w) > V(i-1, w)$  then
15           $L \leftarrow L \cup \{i\}$ ;
16           $w \leftarrow w - w_i$ ;
17         $i \leftarrow i - 1$ ;
18       $S \leftarrow S \cup \{L\}$ ;
19       $k \leftarrow k - 1$ ;
20 return  $S$ 

```

The algorithm starts at $V(n, W)$ respectively and reconstructs a solution $L \subset [n]$, initialized to $L = \emptyset$, bottom-up by making occasional random decisions. Assume the algorithm is at position (i, w) , $1 \leq i \leq n$, $1 \leq w \leq W$. Recall (cf. Algorithm 1) that if $V(i-1, w) < V(i-1, w-w_i) + v_i$ we have no choice and we need to put item i into the knapsack. Likewise, if $V(i-1, w) > V(i-1, w-w_i) + v_i$, item i is guaranteed not to be part of the solution under reconstruction. Thus, in both cases, the decision is deterministic. If $V(i-1, w)$ equals $V(i-1, w-w_i) + v_i$, there are two options how to proceed: in this case with probability

$$\frac{C(i-1, w-w_i)}{C(i, w)}$$

item i is added to L and with the converse probability

$$\frac{C(i-1, w)}{C(i, w)} = 1 - \frac{C(i-1, w-w_i)}{C(i, w)}$$

item i is ignored. If i was packed, the algorithm proceeds (recursively) from $V(i-1, w-w_i)$ and from $V(i-1, w)$ otherwise. This process is iterated while $i > 0$ and $w > 0$. To sample k solutions we may repeat the procedure k times which results in a runtime of $O(kn)$. This is polynomial as long as k is polynomially bounded and of benefit for sampling from an exponential-sized set \mathcal{S}^* if W is

low and hence Algorithm 1 runs in polynomial time. Detailed pseudo-code of the procedure is given in Algorithm 2.

We now show that Algorithm 2 in fact samples each optimal solution uniformly at random from \mathcal{S}^* .

Theorem 2. *Let \mathcal{S}^* be the set of optimal solutions for the knapsack problem and $s \in \mathcal{S}^*$ be an arbitrary optimal solution. Then the probability of sampling s using the sampling approach is $1/|\mathcal{S}^*|$.*

Proof. Let $s \in \mathcal{S}^*$ be an optimal solution. Note that after running Algorithm 1 we have $C(n, W) = |\mathcal{S}^*|$. For convenience we assume that $C(i, w) = 1$ for $w < 0$ to capture the case of invalid solutions. Consider the sequence of $1 \leq r \leq n$ decisions made while traversing back from position (n, W) until a termination criterion is met (either $i \leq 0$ or $w \leq 0$) in Algorithm 2. Let $q_i = \frac{a_i}{b_i}, i \in [r]$ be the decision probabilities in iterations $i \in [r]$. Here, q_i corresponds to q in line 8 of Algorithm 2 if there is a choice whether the corresponding item is taken or not. If there is no choice we can set $q_i = 1 = \frac{x}{x}$ with $x = C(i-1, w)$ if the item is not packed and $x = C(i-1, w - w_i)$ if the item is packed. A key observation is that (1) $b_1 = C(n, W)$, (2) $b_i = a_{i-1}$ holds for $i = 2, \dots, r$ by construction of C and (3) $a_r = 1$ since the termination condition applies after r iterations (see base cases for C in Eq. 4 and Eq. 5). Hence, the probability to obtain s is

$$\begin{aligned} \prod_{i=1}^r q_i &= \frac{a_1}{b_1} \cdot \frac{a_2}{b_2} \cdot \dots \cdot \frac{a_{r-1}}{b_{r-1}} \cdot \frac{a_r}{b_r} \\ &= \frac{a_1}{b_1} \cdot \frac{a_2}{a_1} \cdot \dots \cdot \frac{a_{r-1}}{a_{r-2}} \cdot \frac{1}{a_{r-1}} \\ &= \frac{1}{b_1} \\ &= \frac{1}{C(n, W)} \\ &= \frac{1}{|\mathcal{S}^*|}. \end{aligned}$$

□

Theorem 3. *Let \mathcal{S}^* be the set of optimal solutions for the knapsack problem with n items. Algorithm 2 samples k uniform samples of \mathcal{S}^* in time $O(kn)$.*

Proof. The probabilistic statement follows Theorem 3. For the running time we note that each iteration takes at most n iterations each with a constant number of operations being performed. This is repeated k times which results in $O(kn)$ runtime which completes the proof. □

4 Experiments

In this section we complement the preceding sections with an experimental study and some derived theoretical insights. We first detail the experimental setup, continue with the analysis and close with some remarks.

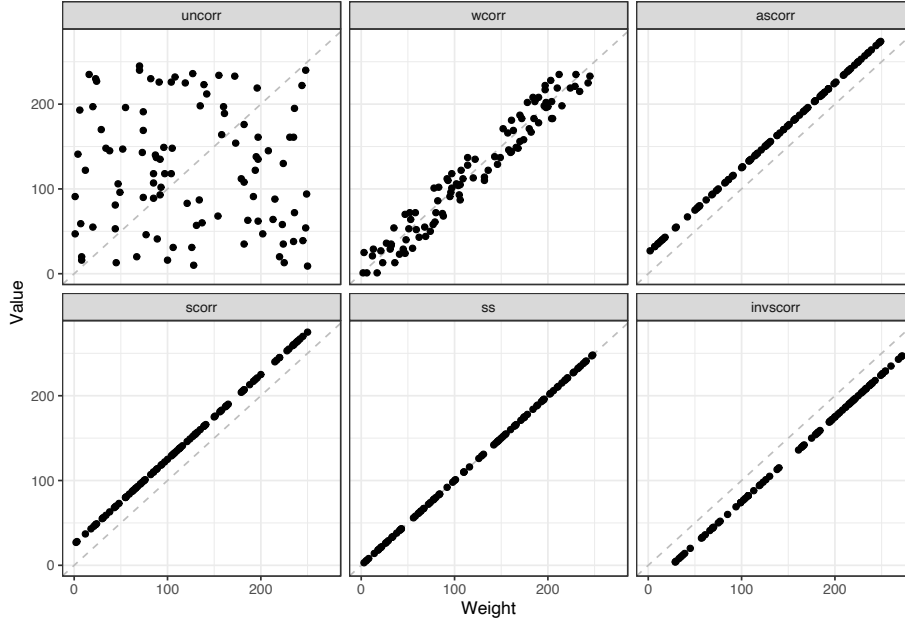


Fig. 1. Showcase of considered instance groups. We show each one instance with $n = 100$ items, $L = 1$ and $R = 250$. The gray dashed diagonal lines serves to aid recognising the differences between correlated instance groups.

4.1 Experimental Setup

The main research question is to get an impression and ideally to understand how the number of global optima develops for classical KP benchmark instances dependent on their generator-parameters. To this end we consider classical random KP benchmark generators as studied, e.g., by Pisinger in his seminal paper on hard knapsack instances [17]. All considered instance groups are generated randomly with item weights w_i sampled uniformly at random within the data range $\{L, \dots, R\}$ with $L = 1$ and varying R for $i \in [n]$; in any case $L < R$. Item profits v_i are in most cases based on a mapping of item weights. The reader may want to take a look at Figure 1 alongside the following description for visual aid.

Uncorrelated (uncorr) Here, both weights and profits are sampled uniformly at random from $\{L, \dots, R\}$.

Weakly correlated (wcorr) Weights w_i are distributed in $\{L, \dots, R\}$ and profits v_i are sampled from $[w_i - R/10, w_i + R/10]$ ensuring $v_i \geq 1$. Here, the values are typically only a small percentage off the weights.

Almost strongly correlated (ascorr) Weights are distributed in $\{L, \dots, R\}$ and v_i are sampled from $[w_i + R/10 - R/500, w_i + R/10 + R/500]$.

Strongly correlated (scorr) Weights w_i are uniformly distributed from the set $\{L, \dots, R\}$ while profits are corresponding to $w_i + R/10$. Here, for all items the profit equals the positive constant plus some fixed additive constant.

Subset sum (susu) In this instance group we have $w_i = v_i \forall i \in [n]$, i.e., the profit equals the weight. This corresponds to strong correlation with additive constant of zero.

Inversely strongly correlated (invscorr) Here, first the profits v_i are sampled from $\{L, \dots, R\}$ and subsequently weights we set $w_i = v_i + R/10$. This is the counterpart of strongly correlated instances.

Correlated instances may seem highly artificial on first sight, but they are of high practical relevance. In economics this situation arises typically if the profit of an investment is directly proportional to the investment plus some fixed charge (strongly correlated) with some problem-dependent noise (weakly / almost strongly correlated).

In our experiments we vary the instance group, the number of items $n \in \{50, 100, \dots, 500\}$ and the upper bound $R \in \{25, 50, 100, 500\}$. In addition, we study different knapsack capacities by setting $D = 11$ and

$$W = \left\lfloor \frac{d}{D+1} \sum_{i=1}^n w_i \right\rfloor$$

for $d = 1, \dots, D$ [16,17]. Intuitively – for most considered instance groups – the number of optima is expected to decrease on average for very low and very high capacities as the number of feasible/optimal combinations is likely to decrease. For each combination of these parameters, we construct 25 random instances and run the DP algorithm to count the number of optima.

Python 3 implementations of the algorithms and generators and the code for running the experiments can be downloaded from a public GitHub repository.³ The experiments were conducted on a MacBook Pro 2018 with a 2,3 GHz Quad-Core Intel Core i5 processor and 16GB RAM. The operating system was macOS Catalina 10.15.6 and python v3.7.5 was used. Random numbers were generated with the built-in python module `random` while `joblist` v0.16.0 served as a multi-core parallelisation backend.

4.2 Insights into the Number of Optima

Figure 2 depicts the distribution of the number of optima for each combination considered in the experiments via boxplots. The data is split row-wise by instance group and col-wise by R (recall that $L = 1$ in any case). Different box-colors indicate the knapsack capacity which – for sake of interpretability – is given in percentage of the total weight, i.e., $\lfloor 100 \cdot (\frac{d}{D+1}) \rfloor$. We observe different patterns dependent on the instance group. For uncorrelated instances (uncorr), we observe only few optima with single outliers reaching $2^8 = 256$ for $R = 50$ and large n .

³ <https://github.com/jakobbossek/LION2021-knapsack-exact-counting>

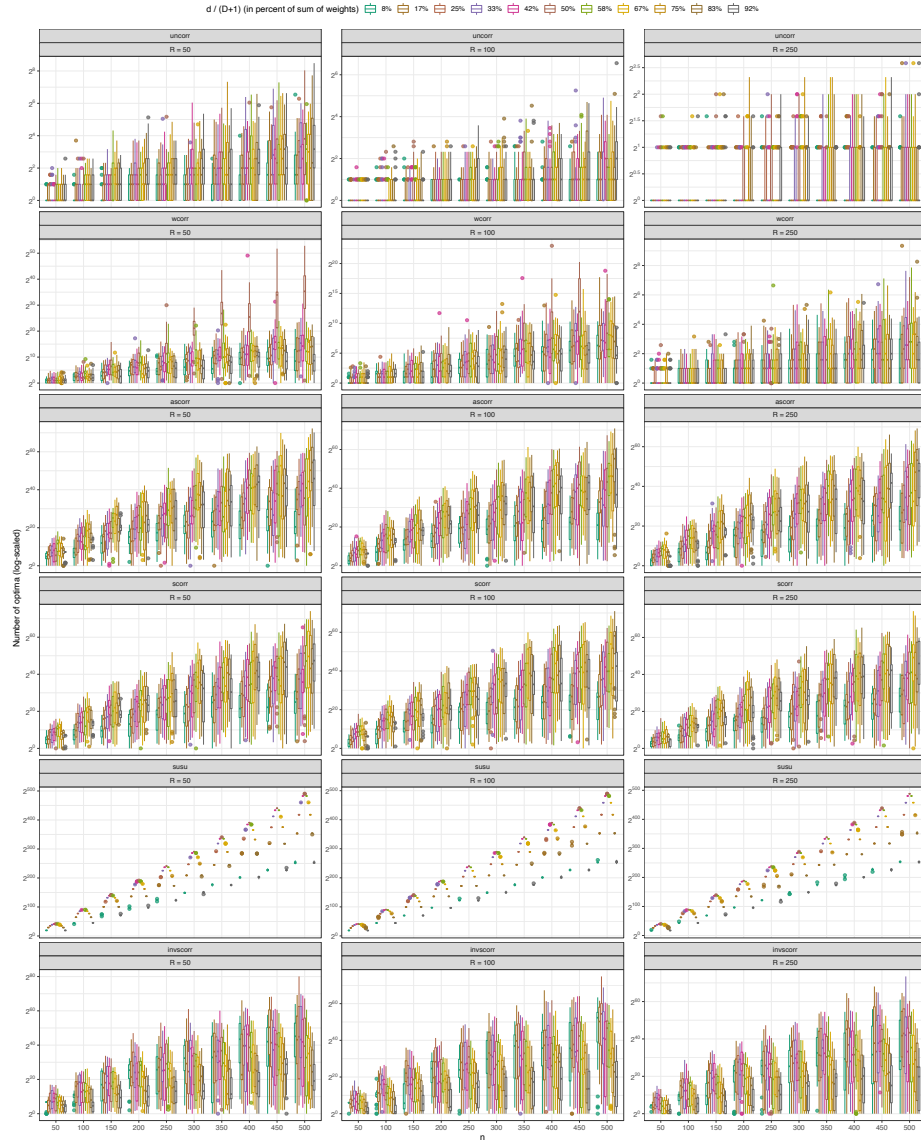


Fig. 2. Boxplots of the number of optima as a function of the number of items n . The data is split by instance group (rows) and the upper bound R (columns). Different colors indicate the knapsack capacity (shown as percentage of the sum of all items weights).

Median values are consistently below $2^4 = 16$. In line with expectation the numbers are highest for relatively small R and high n . In fact, the ratio

$$H = \frac{n}{R}$$

is a good indicator. It is the expected number of elements with weight $w = 1, \dots, R$. In consequence, $H > 1$ and especially $H \gg 1$ indicates many elements with the same weight. In contrast, $H < 1$ indicates that on average there will be at most one element of weight $w = 1, \dots, R$. For all correlated instances, i.e., scorr, ascorr, wcorr, invscorr and susu, we observe a very different pattern. Here, the number of optima grows exponentially with growing n given a fixed upper bound R . Even if H is low there is huge number of optima. By far the highest count of optima can be observed for subset sum (susu) instances where even peaks with up to $\approx 3\%$ of all 2^n solutions are optimal. Here, the boxplots look degenerate, because the variance is very low. Recall that for this type of instance we have $w_i = v_i \forall i \in [n]$ and thus for each solution s the equality $w(s) = v(s)$ holds. In consequence we aim to maximally exploit the knapsack capacity.

To get a better understanding we consider a subset-sum type knapsack instance with $w_i \in \{1, \dots, R\}, w_i = v_i, \forall i \in [n]$. Assume for ease of calculations that n is a multiple of R and there are exactly (n/R) items of each weight $w \in \{1, \dots, R\}$, i.e., $|\{i \in [n] \mid w_i = w\}| = n/R$. Note that this corresponds to the expected number of w -weights if n such weights are sampled uniformly at random from $\{1, \dots, R\}$. Consider $W = \frac{1}{2} \sum_{i=1}^n w_i$. Recall that given this instance, one way we can build an optimum $s \subset [n]$ with $w(s) = v(s) = W$ is by choosing each $(\frac{1}{2}) \cdot (\frac{n}{R})$ items from each weight class, i.e., half of these items (note that there are many more combinations leading to profit W). With this we get

$$|\mathcal{S}^*| \geq \left(\frac{\frac{n}{R}}{\frac{n}{2R}}\right)^R \geq \left(\left(\frac{n}{R} \cdot \frac{2R}{n}\right)^{\frac{n}{2R}}\right)^R = 2^{\frac{n}{2}}.$$

Here we used to the well-known lower bound $\binom{n}{k} \geq \left(\frac{n}{k}\right)^k$ for the binomial coefficient. This simple bound establishes that we can expect at least $2^{n/2}$ optima for subset-sum instances if the capacity is set accordingly.

With respect to the knapsack capacity Figure 2 also reveals different patterns. For inverse strongly correlated instances we observe a decreasing trend with increasing capacity. The vice versa holds for weakly, almost strongly and strongly correlated instances. This is in line with intuition as the size of the feasible search space also grows significantly.

However, note that in general the knapsack capacity can have a massive effect on the number of optima.

Theorem 4. *For every even n there exist a KP instance and a weight capacity W such that $|\mathcal{S}^*|$ is exponential, but $|\mathcal{S}^*| = 1$ for $W' = W + 1$.*

Proof. Consider an instance with n items (n even), where $w_i = v_i = 1$ for $i \in [n-1]$ and $w_n = \frac{n}{2} + 1$. Let $v_n > \frac{n}{2} + 1$. Now consider the knapsack capacity

$W = \frac{n}{2}$. Then every subset of $\frac{n}{2}$ items from the first $n - 1$ items is optimal with total weight W and total value W while the n -th item does not fit into the knapsack. There are at least

$$\begin{aligned}
\binom{n-1}{\frac{n}{2}} &= \frac{(n-1)!}{(n/2)!(n-1-n/2)!} \\
&\geq \frac{(n-1)!}{\left(\left(\frac{n}{2}\right)!\right)^2} \\
&\geq \frac{\sqrt{2\pi}\sqrt{n-1}(n-1)^{n-1}2^n e^n}{2e^{n-1}4(2\pi)n^{n+1}} \\
&= \frac{2^{n-3}e}{\sqrt{2\pi}} \cdot \underbrace{\left(\frac{n-1}{n}\right)^{n-1}}_{\geq e^{-1}} \cdot \frac{\sqrt{n}}{n^2} \cdot \underbrace{\sqrt{1-\frac{1}{n}}}_{\geq 1/\sqrt{2}} \\
&\geq \frac{1}{\sqrt{\pi}} \cdot \frac{2^{n-4}}{n^{3/2}} \\
&= \Omega(2^{n-4}/n^{3/2})
\end{aligned}$$

optima in this case. Here we basically used Stirling's formula to lower/upper bound the factorial expressions to obtain an exponential lower bound. Now instead consider the capacity $W' = W + 1$. The n -th item now fits into the knapsack which results in a unique optimum with weight $W' = \frac{n}{2} + 1$ and value $v_n > \frac{n}{2} + 1$ which cannot be achieved by any subset of light-weight items. \square

4.3 Closing Remarks

Knapsack instances with correlations between weights and profits are of high practical interest as they arise in many fixed charge problems, e.g., investment planning. In this type of instances item profits correspond to their weight plus/minus a fixed or random constant. Our experimental study suggests an exponential increase in the number of global optima for such instances which justifies the study and relevance of the considered counting and sampling problem.

5 Conclusion

We considered the problem of counting exactly the number of optimal solutions of the zero-one knapsack problem. We build upon the classic dynamic programming algorithm for the optimisation version. Our modifications allow to solve the counting problem in pseudo-polynomial runtime complexity. Furthermore, we show how to sample uniformly at random from the set of optimal solutions without explicit construction of the whole set. Computational experiments and derived theoretical insights reveal that for variants of problem instances with correlated weights and profits (a group which is highly relevant in real-world

scenarios) and for a wide range of problem generator parameters, the number of optimal solutions can grow exponentially with the number of items. These observations support the relevance of the considered counting and sampling problems.

Future work will focus on (approximate and exact) counting/sampling of high-quality knapsack solutions which all fulfill a given non-optimal quality threshold. In addition, in particular if the set of optima has exponential size, it is desirable to provide the decision maker with a diverse set of high-quality solutions. Even though the introduced sampling is likely to produce duplicate-free samples if the number of solutions is exponential, it seems more promising to bias the sampling process towards a diverse subset of optima, e.g., with respect to item-overlap or entropy. This opens a whole new avenue for upcoming investigations.

References

1. Broder, A.Z., Mayr, E.W.: Counting minimum weight spanning trees. *Journal of Algorithms* **24**(1), 171–176 (Jul 1997)
2. Cai, J.Y., Chen, X.: *Complexity Dichotomies for Counting Problems: Volume 1, Boolean Domain*. Cambridge University Press, USA, 1st edn. (2017)
3. Do, A.V., Bossek, J., Neumann, A., Neumann, F.: Evolving diverse sets of tours for the travelling salesperson problem. In: *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2020*. pp. 681–689. ACM (2020)
4. Dyer, M.: Approximate counting by dynamic programming. In: *Proceedings of the Thirty-Fifth Annual ACM Symposium on Theory of Computing*. p. 693–699. STOC '03, Association for Computing Machinery, New York, NY, USA (2003)
5. Dyer, M.E., Goldberg, L.A., Greenhill, C.S., Jerrum, M.: On the relative complexity of approximate counting problems. In: *Proceedings of the International Workshop on Approximation Algorithms for Combinatorial Optimization (APPROX)*. *Lecture Notes in Computer Science*, vol. 1913, pp. 108–119. Springer (2000)
6. Fichte, J.K., Hecher, M., Meier, A.: Counting complexity for reasoning in abstract argumentation. In: *Proceedings of the The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019*. pp. 2827–2834. AAAI Press (2019)
7. Fichte, J.K., Hecher, M., Morak, M., Woltran, S.: Exploiting treewidth for projected model counting and its limits. In: *Proceedings of the 21st International Conference Theory and Applications of Satisfiability Testing, SAT 2018*. *Lecture Notes in Computer Science*, vol. 10929, pp. 165–184. Springer (2018)
8. Fournier, H., Malod, G., Mengel, S.: Monomials in arithmetic circuits: Complete problems in the counting hierarchy. *Computational Complexity* **24**(1), 1–30 (2015)
9. Katz, M., Sohrabi, S.: Reshaping diverse planning. In: *Proceedings of the The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020*. pp. 9892–9899. AAAI Press (2020)
10. Katz, M., Sohrabi, S., Udrea, O.: Top-quality planning: Finding practically useful sets of best plans. In: *Proceedings of the The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020*. pp. 9900–9907. AAAI Press (2020)
11. Katz, M., Sohrabi, S., Udrea, O., Winterer, D.: A novel iterative approach to top- k planning. In: *Proceedings of the Twenty-Eighth International Conference on Automated Planning and Scheduling, ICAPS 2018*. pp. 132–140. AAAI Press (2018)

12. Kellerer, H., Pferschy, U., Pisinger, D.: Knapsack Problems. Springer, Berlin, Germany (2004)
13. Mihalák, M., Šrámek, R., Widmayer, P.: Approximately counting approximately-shortest paths in directed acyclic graphs. *Theory of Computing Systems* **58**(1), 45–59 (Jan 2016)
14. Neumann, A., Gao, W., Doerr, C., Neumann, F., Wagner, M.: Discrepancy-based evolutionary diversity optimization. In: Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2018. pp. 991–998 (2018)
15. Neumann, A., Gao, W., Wagner, M., Neumann, F.: Evolutionary diversity optimization using multi-objective indicators. In: Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2019. pp. 837–845 (2019)
16. Pisinger, D.: Core problems in knapsack algorithms. *Operations Research* **47**(4), 570–575 (Apr 1999)
17. Pisinger, D.: Where are the hard knapsack problems? *Computers & Operations Research* **32**(9), 2271 – 2284 (2005)
18. Rizzi, R., Tomescu, A.I.: Faster FPTASes for counting and random generation of knapsack solutions. *Information and Computation* **267**, 135–144 (2019)
19. Sohrabi, S., Riabov, A.V., Udrea, O., Hassanzadeh, O.: Finding diverse high-quality plans for hypothesis generation. In: Proceedings of the 22nd European Conference on Artificial Intelligence, ECAI 2016. *Frontiers in Artificial Intelligence and Applications*, vol. 285, pp. 1581–1582. IOS Press (2016)
20. Ulrich, T., Thiele, L.: Maximizing population diversity in single-objective optimization. In: Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2011. pp. 641–648 (2011)
21. Valiant, L.G.: The complexity of computing the permanent. *Theoretical Computer Science* **8**, 189–201 (1979)
22. Štefankovič, D., Vempala, S., Vigoda, E.: A deterministic polynomial-time approximation scheme for counting knapsack solutions. *SIAM Journal on Computing* **41**(2), 356–366 (Apr 2012)