European Journal of Operational Research 301 (2022) 841–854

Contents lists available at ScienceDirect

European Journal of Operational Research

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

Discrete Optimization

A new class of hard problem instances for the 0–1 knapsack problem

Jorik Jooken[*], Pieter Leyman, Patrick De Causmaecker

Department of Computer Science, KU Leuven Kulak, Etienne Sabbelaan 53, Kortrijk 8500, Belgium

ARTICLE INFO

Article history:
Received 24 June 2021
Accepted 4 December 2021
Available online 9 December 2021

Keywords:
Combinatorial optimization
0–1 knapsack problem
Problem instance hardness

ABSTRACT

The 0–1 knapsack problem is an important optimization problem, because it arises as a special case of a wide variety of optimization problems and has been generalized in several ways. Decades of research have resulted in very powerful algorithms that can solve large knapsack problem instances involving thousands of decision variables in a short amount of time. Current problem instances in the literature no longer challenge these algorithms. However, hard problem instances are important to demonstrate the strengths and weaknesses of algorithms and this knowledge can in turn be used to create better performing algorithms. In this paper, we propose a new class of hard problem instances for the 0–1 knapsack problem and provide theoretical support that helps explain why these problem instances are hard to solve to optimality. A large dataset of 3240 hard problem instances was generated and subsequently solved on a supercomputer, using approximately 810 CPU-hours. The analysis of the obtained results shows to which extent different parameters influence the hardness of the problem instances. This analysis also demonstrates that the proposed problem instances are a lot harder than the previously known hardest instances, despite being much smaller.

1. Introduction

Many interesting combinatorial optimization problems are NP-hard. For such problems, it is not known whether algorithms exist that can solve[1] them in polynomial time. In fact, almost all researchers believe that such algorithms do not exist (Gasarch, 2019). However, this does not rule out the possibility that most problem instances could potentially be solved very quickly. Indeed, for several NP-hard problems there are many problem instances from the literature that have already been solved in a reasonable amount of time. For some problems, the sizes of the problem instances that have been solved are small to moderate (e.g. for the quadratic assignment problem Lawler, 1963; Loiola, de Abreu, Boaventura-Netto, Hahn, & Querido, 2007), whereas for other problems even very large problem instances involving thousands of decision variables have been solved (e.g. for a weakly NP-hard problem like the 0–1 knapsack problem Martello, Pisinger, & Toth, 1999; Pisinger, 2005, but also for strongly NP-hard problems like the travelling salesman problem Applegate, Bixby, Chvátal, & Cook, 2006; Applegate et al., 2009 and the Hamiltonian completion problem Jooken, Leyman, & De Causmaecker, 2020a). Especially this last category of problems is remarkable and the algorithms used to solve these problems are often the result of years of research focused on finding algorithmic improvements. For the 0–1 knapsack problem, the ***Combo*** algorithm (Martello et al., 1999) is such an algorithm which can solve many problem instances involving thousands of items in a matter of (milli)seconds. For this reason, the 0–1 knapsack problem is often considered to be an easy NP-hard problem (Pisinger, 2005), notwithstanding the fact that it has been challenging in the past for many years to solve large knapsack problem instances.

When such powerful algorithms exist, it is not always an easy task to create benchmarks that challenge these algorithms, because it is not always known how to create hard problem instances. Furthermore, it is not always clear which problem instance features, apart from the size, influence their hardness. Another reason why it can be difficult to create such problem instances, is because these hard problem instances often must obey certain combinations of constraints that are difficult to fulfill at the same time. Despite this challenge, such problem instances are important because these provide insights into the weaknesses and strengths of certain algorithms and also provide valuable insights into the problem as a whole. This knowledge can in turn be used to design more efficient optimization approaches. This is precisely one of the goals of benchmarks and thus such hard problem instances should ideally

* Corresponding author.
E-mail addresses: jorik.jooken@kuleuven.be (J. Jooken), pieter.leyman@kuleuven.be (P. Leyman), patrick.decausmaecker@kuleuven.be (P. De Causmaecker).

[1] In this paper, we will use the term “solve” to indicate that an algorithm finds at least one of the optimal solutions.

https://doi.org/10.1016/j.ejor.2021.12.009
0377-2217/© 2021 Elsevier B.V. All rights reserved.

be contained in benchmarks. In the current benchmarking landscape, such hard problem instances are unfortunately not always included.

For the 0–1 knapsack problem, several classes of hard problem instances are known. For example, Chvátal (1980) proposed a class of problem instances for which he was able to prove that the running time of all algorithms with certain specific properties must be at least exponential in the number of items. These problem instances are mainly of theoretical importance, but cannot be tested on actual implementations for solving the 0–1 knapsack problem, because the number of digits required to represent the integer coefficients is also of the same order of magnitude as the number of items and most implementations for solving the 0–1 knapsack problem only support 32-bit or 64-bit integers. The problem instances proposed by Pisinger (2005) in “Where are the hard knapsack problems?” form another well known example of hard problem instances. In contrast with the instances of Chvátal, Pisinger's instances are more practical and were empirically shown to be hard to solve for exact algorithms for the 0–1 knapsack problem. In the current paper, we build upon this line of research concerned with hard problem instances for the 0–1 knapsack problem.

The main contributions of this paper are as follows:

1. We introduce a new class of hard problem instances for the 0–1 knapsack problem for which two theorems are proven that help characterize why these problem instances are hard. These theorems provide further insight into solutions that are hard to prune from the search space and the structure of optimal solutions. Since the 0–1 knapsack problem is a fundamental problem and it is a special case of several packing related problems (Bonyadi, Michalewicz, & Barone, 2013; Kellerer, Pferschy, & Pisinger, 2004b; Pferschy & Schauer, 2009), this work is also of wider interest.
2. A large dataset of 3240 hard problem instances of the proposed class was systematically generated using a full factorial design of experiments. A big computational effort was made to solve these instances on a set of 20 nodes in a supercomputer center, needing approximately 810 CPU-hours to complete all experiments. We envision two ways in which this dataset can be useful. The first use of this dataset is for researchers in the cutting and packing community who are interested in the 0–1 knapsack problem. Since, in our opinion, good benchmarks must contain a wide variety of problem instance classes, we propose to use our dataset as part of a larger benchmark set of problem instances containing several problem instance classes (e.g. the instances of Pisinger, 2005). Most problem instances in the dataset that we propose are very challenging. The dataset has been made publicly available at https://github.com/JorikJooken/knapsackProblemInstances, making it easy for anybody who wishes to integrate our dataset in their own benchmarks. Secondly, this dataset can be used by researchers in the field of algorithm runtime prediction (see e.g. Hutter, Xu, Hoos, & Leyton-Brown, 2014), where the goal is to predict the runtime of an algorithm based on problem instance features. To make this dataset more widely useful outside of the cutting and packing community, the problem instances have been characterized in terms of the parameters that were used to generate them and the runtimes and optimal solutions of all problem instances have been recorded such that computationally expensive algorithm runs do not have to be repeated.
3. The analysis of the computational experiments provides insight into how different parameters of the proposed problem instance generator influence the hardness of the problem instances. This analysis also reveals that the proposed problem instances are several orders of magnitude harder than the previously hardest problem instances, despite being much smaller than those instances. The hardest problem instances from this paper require 60 times more CPU time than the previously hardest problem instances. Hence, this class of problem instances expands the knowledge of the cutting and packing community and poses a new challenge for practitioners.

The rest of this paper is structured as follows: in Section 2 a formal description of the 0–1 knapsack problem is given. An overview of relevant literature and related work is given in Section 3. The proposed hard class of problem instances for the 0–1 knapsack problem is described in Section 4. We discuss several theoretical properties of this hard class of problem instances in Section 5. The experimental setup and the analysis of the obtained results will be discussed in Section 6. Finally, the key points of this paper and further challenges for future work will be given in Section 7.

2. Problem description

The 0–1 knapsack problem is a classical NP-hard optimization problem (Kellerer, Pferschy, & Pisinger, 2004a). In this problem, one is given a knapsack with an integer capacity c and a set of n items, which each have an integer profit p_i and an integer weight w_i. The goal is to select a subset of items to put into the knapsack such that the total value is maximized and the total weight does not exceed the knapsack capacity. This problem can be written as the following optimization problem:

$$\text{maximize} \quad \sum_{i=1}^{n} p_i x_i \tag{1a}$$

$$\text{subject to} \quad \sum_{i=1}^{n} w_i x_i \le c \tag{1b}$$

$$x_i \in \{0, 1\}, \quad \forall i \in \{1, 2, \dots, n\} \tag{1c}$$

Here, the problem variables are the knapsack capacity c, the number of items n and their associated weights w_i $(i = 1, \dots, n)$ and profits p_i $(i = 1, \dots, n)$. The n binary decision variables x_i $(i = 1, \dots, n)$ indicate whether item i should be put into the knapsack. The objective function (1a) represents the total value of the selected items, whereas constraint (1b) represents the capacity constraint and constraints (1c) impose the decision variables to be binary variables. In what follows, we will assume the weights and profits to be strictly positive. We do not lose generality through this assumption, because the other cases can be transformed to an equivalent problem instance with only strictly positive weights and profits (Kellerer et al., 2004a). Finally, to avoid trivial cases it is also assumed that $\sum_{i=1}^{n} w_i > c$ and that $w_i \le c$ $(i = 1, \dots, n)$.

3. Literature overview

The purpose of this section is to give an overview of relevant research for the 0–1 knapsack problem that helps to provide a context for the results in this paper. Hence, it is not exhaustive, but rather several important topics are discussed in more detail. For a more complete overview, the interested reader is referred to Kellerer et al. (2004a) and Pisinger & Toth (1998).

Although the 0–1 knapsack problem is NP-hard, it is one of the easier problems within this complexity class. It can be solved in pseudopolynomial time with a time complexity of $O(nc)$ (Bellman, 1966). Additionally, several (fully) polynomial time approximation schemes exist (Caprara, Kellerer, Pferschy, & Pisinger, 2000; Kellerer & Pferschy, 1999; 2004). However, not all of these algorithms are necessarily practical and may lead to large running times. A lot of attention has been devoted in the literature to developing algorithms that have a good performance on a large number of problem instances. Most successful algorithms rely on either the principles of branch-and-bound, dynamic programming or a hybrid of both, in which dynamic programming states that cannot lead to optimal solutions are fathomed based on bounds. Another crucial concept that is used by these algorithms, is the concept of *cores*. It has been observed for many problem instances that the structure of the optimal solution often relies on only a small subset of items, which form the core (Kellerer et al., 2004a). There are several variations for the precise definition of what a core is and most of them require the optimal solution to be known in order to determine the exact core. In this paper, we will use the definition by Balas & Zemel (1980): if we define the rank of an item as the index in the list that is obtained by sorting the items in decreasing order of their profit-weight ratio $\frac{p_i}{w_i}$, then the core consists of all items with a rank between the minimum and maximum rank of an item for which the decision variable x_i has a different value in the optimal solution to the 0–1 knapsack problem and the optimal solution to its linear relaxation. Since the optimal solution is not known before solving the problem instance, there are several strategies to deal with finding the core.

Martello & Toth (1977) proposed ***MT1***. This is a branch-and-bound algorithm for the 0–1 knapsack problem, using a new upper bound which improved the upper bound from Dantzig (1957). Martello and Toth later improved ***MT1*** by proposing ***MT2*** (Martello & Toth, 1988). In this algorithm, they used a stronger upper bound and integrated the concept of cores, by first determining an approximate core of a fixed size, which is then solved by ***MT1***. Pisinger proposed another branch-and-bound algorithm, called ***Expknap*** (Pisinger, 1995), in which the size of the core gradually expands (i.e. an expanding core). Pisinger later also proposed ***Minknap*** (Pisinger, 1997), an algorithm based on dynamic programming for which unpromising states are fathomed. Pisinger was able to prove that this algorithm enumerates the smallest possible core. Finally, Martello et al. (1999) proposed ***Combo*** in 1999, which is based on a combination of various ideas. It has great similarities with ***Minknap***, but also uses cardinality constraints which are surrogate relaxed to the original problem. Several algorithmic ideas are gradually introduced in ***Combo***, depending on the number of encountered dynamic programming states. Although ***Combo*** was introduced more than twenty years ago, it still represents the current state-of-the-art. This is not surprising, considering that it was the result of more than two decades of research and the results that it obtained left little room for improvement. ***Combo*** can solve most instances from the literature (even the largest ones containing several thousands of items) in a matter of (milli)seconds. It has for example been observed that frequently the sorting time of $O(n\log(n))$ (to sort the items in decreasing order of their profit-weight ratio) is the dominant term (Kellerer et al., 2004a; Pisinger, 1997), and thus this algorithm is very fast. This is an impressive achievement, considering that the 0–1 knapsack problem is NP-hard. It also illustrates the difficulty that researchers have experienced to find hard problem instances that could challenge ***Combo***, because it is such a powerful algorithm. Later in Section 6, it will become clear that the problem instances from this paper fill this gap.

There are several papers in the literature that specifically focus on hard problem instances for the 0–1 knapsack problem. Most of these hard problem instances are mainly of theoretical importance, with the instances of Pisinger (2005) and Smith-Miles, Christiansen, & Muñoz (2021) as important exceptions. Unlike the instances from Pisinger (2005) and Smith-Miles et al. (2021), those theoretically important instances cannot be tested on actual implementations for solving the 0–1 knapsack problem, because the number of digits required to represent the integer coefficients is at least of the same order of magnitude as the number of items n, whereas most implementations for solving the 0–1 knapsack problem only support 32-bit or 64-bit integers. Chvátal (1980) proposed a class of problem instances for which he was able to prove that the running time of all algorithms with certain specific properties must be at least exponential in n. Chvátal's results were later also extended by Gu, Nemhauser, & Savelsbergh (1999) and Jukna & Schnitger (2011). These extensions differ in the assumptions that they make about the power of the algorithms that are used to solve the problem instances. Gu et al. (1999) study algorithms based on using lifted cover inequalities, whereas Jukna & Schnitger (2011) allow the algorithms to use memoization and strong bounds. The sizes of the coefficients considered in these three papers are all extremely large. The instances proposed by Pisinger (2005) in "Where are the hard knapsack problems?" are more practical. Pisinger proposed 13 different classes of problem instances and has empirically shown that several of these problem instances are hard to solve for exact knapsack solvers. In contrast with the three previous papers, no theoretical guarantees are given for Pisinger's instances, but rather the actual performance of existing algorithms is considered. Hence, Pisinger's work is the closest resemblance from the literature to the current paper, although in contrast with Pisinger (2005) our instances allow us to provide both theoretical and empirical results (see Sections 5 and 6 respectively). Recently, Smith-Miles et al. (2021) introduced a set of problem instances that is even more diverse than the already diverse set of instances from Pisinger (2005). They identify different regions in the problem instance space and manually determine a set of target points, representing combinations of features of problem instances that are not present in existing datasets. Smith-Miles et al. (2021) state that they were mainly interested in "filling 'holes' in the instance space or pushing beyond the outer boundary of the existing instances in directions which appear to correspond to more difficult instances". Hence, the primary objective of Smith-Miles et al. (2021) was to generate more diverse problem instances. After determining the set of target points, they use genetic algorithms to evolve a new set of problem instances with an objective function that attempts to minimize the distance (in terms of a set of features) between the newly created problem instances and the target points. As we will later show, while these instances add the intended diversity to available benchmark instances, they are not necessarily harder.

4. Problem instance generator

The problem instance generator that we propose is a stochastic generator that expects seven parameters (n, c, g, f, ϵ, s and b) as input and will produce a problem instance for the 0–1 knapsack problem as output. Here, the parameters n, c, g, s and b should be positive integers, whereas f and ϵ should be positive real numbers. Additionally, b should be greater than 1. Many different combinations of parameter values will yield valid problem instances and in principle any of those combinations could be used to generate problem instances. However, by appropriately choosing the parameters we obtain several nice theoretical properties. These properties influence for example the structure of optimal solutions and the existence of solutions that are hard to prune from the search space. Hence, in the rest of this paper we will assume several other relationships between the parameters that reflect the intended use of the problem instance generator. These relationships will be made explicit in Section 5 when the properties will be discussed. For now, however, we will explain the problem instance generator in its full generality without going into too much detail about the expected relationships between the parameters. The problem instances proposed in this paper can also be seen as a more powerful generalization of problem instances that have been proposed by us before (Jooken, Leyman, De Causmaecker, & Wauters, 2020b).

Before we give the precise description of the proposed problem instances, we attempt to give some intuition. The problem instances from this paper consist of items that can be partitioned into several groups with exponentially decreasing profits and weights. Items within the same group have similar profits and weights, but are slight perturbations of each other. The last group differs from the other groups and consists of items with uncorrelated small profits and weights. This last group does not significantly affect the value of the optimal solution, but introduces items with a large variety of profit-weight ratios, which can potentially lead to large cores. The parameters of the proposed problem instance generator affect several properties from this paragraph such as the number of groups, the relative size of the groups, the degree of perturbation and the ratio between the exponentially decreasing profits and weights.

More specifically, the generator will produce a problem instance for which there are n items and a knapsack with capacity c. Every item belongs to one of g different groups of items. The parameter f influences the number of items in the different groups and it represents (at least approximately) the fraction of items that belong to the last group. The remaining items are equally distributed over the first $g-1$ groups. We define m as the number of items in group i ($1 \leq i \leq g-1$). We write “approximately”, because the situation is a bit more complicated, since nf is not necessarily an integer and $n(1-f)$ is not necessarily divisible by $g-1$. To be more precise, the first $g-1$ groups each contain exactly $m = \lfloor \frac{n-\lfloor nf \rfloor}{g-1} \rfloor$ items, whereas the last group contains the remaining $n-(g-1)m$ items. This last expression is indeed approximately equal to nf, as desired. The items will be placed one by one in consecutive groups until the groups contain the desired number of items. For every item j ($1 \leq j \leq n$) two small integers $r_{1,j}$ and $r_{2,j}$ are sampled uniformly at random between 1 and s. If item j belongs to group i (with $1 \leq i \leq g-1$), its profit p_j and weight w_j are set to $p_j = \left\lfloor \left(\frac{1}{b^i}+\epsilon\right)c \right\rfloor + r_{1,j}$ and $w_j = \left\lfloor \left(\frac{1}{b^i}+\epsilon\right)c \right\rfloor + r_{2,j}$. Finally, if item j belongs to the last group, its profit p_j and weight w_j are set to $p_j = r_{1,j}$ and $w_j = r_{2,j}$. By defining the problem instances in this way, several nice theoretical properties can be proven. This will be the topic of the next section.

In the rest of this paper, we will refer to these problem instances as noisy multi-group exponential (NMGE) problem instances. This name is based on the fact that the items belong to multiple groups, where the profits and weights of items amongst the first $g-1$ groups are exponentially decreasing with approximately a factor of b. The parameter ϵ, the floor function and the randomly generated integers $r_{1,j}$ and $r_{2,j}$ ($1 \leq j \leq n$) are the reason why the factor is not exactly equal to b and can be interpreted as a kind of noise.

5. Theoretical properties

The theoretical properties in this section give further insight into the structure of the problem instances and help us to explain why these are hard to solve. We will state, prove and discuss these properties.

5.1. Near-optimal solutions

The NMGE problem instances from this paper were constructed such that there are many feasible solutions for which the objective function value is relatively close to the optimal objective function value. This makes it hard for algorithms to prune such solutions from the search space based on calculating lower and upper bounds. Hence, the existence of such solutions might make algorithms for the 0–1 knapsack problem slow, because large parts of the search space must be explored to guarantee finding an optimal solution. In this subsection, we will prove that all *inclusionwise maximal* solutions have an objective function value relatively close to the optimal objective function value. We define the notion of an *inclusionwise maximal* solution in the following definition:

Definition 1. Consider a problem instance ω consisting of a knapsack with capacity c and n items with profits $p_{\omega,1}, p_{\omega,2}, \ldots, p_{\omega,n}$ and weights $w_{\omega,1}, w_{\omega,2}, \ldots, w_{\omega,n}$. A solution $x_{\omega,1}, x_{\omega,2}, \ldots, x_{\omega,n}$ is called inclusionwise maximal relative to ω if it is a feasible solution (i.e. $\sum_{i=1}^{n} w_{\omega,i}x_{\omega,i} \leq c$) and there does not exist an item j that can be added to the knapsack without violating the capacity constraint (i.e. $\nexists j : x_{\omega,j} = 0 \wedge w_{\omega,j} + \sum_{i=1}^{n} w_{\omega,i}x_{\omega,i} \leq c$).

Note that it is easy to find an inclusionwise maximal solution (e.g. by greedily adding items to the knapsack as long as the capacity constraint is not violated). Hence, for the problem instances that we propose in this paper, it is easy to find near-optimal solutions, but hard to find optimal solutions. Any optimal solution has to be an inclusionwise maximal solution, because the profits of the items are assumed to be positive integers. In this subsection, we will prove that inclusionwise maximal solutions are indeed near-optimal. We will do this by proving a lower bound for the worst-case performance ratio between the objective function value of an inclusionwise maximal solution and an optimal solution (Theorem 1).

The generated problem instances depend on the parameters of the problem instance generator (n, c, g, f, ϵ, s and b) and the random values that are generated for the profit and weight of each item. We will denote by $\Omega(n, c, g, f, \epsilon, s, b)$ the set of all problem instances that can be generated by the problem instance generator using the given parameters. We will be interested in the worst-case performance ratio between the objective function value of an inclusionwise maximal solution and an optimal solution over the problem instances from this parameterized set. Note that the set $\Omega(n, c, g, f, \epsilon, s, b)$ is finite. It contains precisely s^{2n} problem instances, because the problem instance generator independently generates a random integer between 1 and s for the profit and weight of each item. Before we can prove Theorem 1, we first need to prove three technical lemmas. Since there are quite a lot of different parameters involved in the problem instance generator, we added the description of several commonly occurring expressions in Table 1 to make it easier for the readers to follow these lemmas. Some of these expressions will only be defined later in the paper, but are already mentioned in this table.

The first lemma characterizes for every problem instance certain specific groups for which the items cannot all be contained in the knapsack at the same time.

Lemma 1. *Let $\omega \in \Omega(n, c, g, f, \epsilon, s, b)$ be a problem instance and $x_{\omega,1}, x_{\omega,2}, \ldots, x_{\omega,n}$ be an inclusionwise maximal solution relative to ω. If d is an integer such that $1 \leq d \leq \min(g-1, \lfloor \log_b(m) \rfloor)$, then there exists an item j in group d which is not included in the knapsack (i.e. $x_{\omega,j} = 0$).*

Proof. Recall from the description of the problem instance generator that the first $g-1$ groups of items each contain exactly $m = \lfloor \frac{n-\lfloor nf \rfloor}{g-1} \rfloor$ items, and item j in group i has a weight of $\lfloor (\frac{1}{b^i} + \epsilon)c \rfloor + r_{2,j}$ for some integer $r_{2,j}$ between 1 and s. For ease of notation, we define D as the set of all indices of items in group d. Including all items of group d in the knapsack would violate the capacity constraint, because

$$\sum_{k \in D} w_{\omega,k} \geq \left(\left\lfloor \left(\frac{1}{b^d}+\epsilon\right)c \right\rfloor + 1 \right) m > \left(\left(\frac{1}{b^d}+\epsilon\right)c - 1 + 1 \right) m$$
$$\geq c\frac{m}{b^d} \geq c\frac{m}{b^{\log_b(m)}} = c$$

Hence, there must be some $j \in D$ such that $x_{\omega,j} = 0$, because otherwise $\sum_{k \in D} x_{\omega,k} w_{\omega,k} > c$. □

Table 1
Overview of commonly occurring expressions.

Expression	Description
n	The number of items of a problem instance
c	The capacity of the knapsack
g	The number of groups of items of a problem instance
f	The approximate fraction of items in the last group (group g)
ϵ	A noise parameter
s	An upper bound for profits and weights of items in group g
b	The base of the exponent used for the exponentially decreasing profits and weights
$\Omega(n,c,g,f,\epsilon,s,b)$	The set of all problem instances that can be generated using the given parameters
ω	A problem instance from the set $\Omega(n,c,g,f,\epsilon,s,b)$
$p_{\omega,j}$	The profit of item j for problem instance ω
$w_{\omega,j}$	The weight of item j for problem instance ω
$x_{\omega,j}$	A binary decision variable indicating whether item j from ω should be included into the knapsack
$r_{1,j}, r_{2,j}$	Two randomly generated integers between 1 and s for item j
$p_{\omega,j} = \left\lfloor \left(\frac{1}{b^i}+\epsilon\right)c \right\rfloor + r_{1,j}$	The profit of item j from ω in case it belongs to group i (for $1 \le i \le g-1$)
$p_{\omega,j} = r_{1,j}$	The profit of item j from ω in case it belongs to group g
$w_{\omega,j} = \left\lfloor \left(\frac{1}{b^i}+\epsilon\right)c \right\rfloor + r_{2,j}$	The weight of item j from ω in case it belongs to group i (for $1 \le i \le g-1$)
$w_{\omega,j} = r_{2,j}$	The weight of item j from ω in case it belongs to group g
$m = \lfloor \frac{n-\lfloor nf \rfloor}{g-1} \rfloor$	The exact number of items in group i (for $1 \le i \le g-1$)
$n-(g-1)m$	The exact number of items in group g
d	At least one item from group d cannot be included into the knapsack for any inclusionwise maximal solution
D	The set of all indices of items in group d
F	The set of all indices of items in the first $g-1$ groups
L	The set of all indices of items in the last group (group g)
α, β	The objective function value of any inclusionwise maximal solution is at least as large as $\alpha(c-\beta)$
γ, δ	The objective function value of any inclusionwise maximal solution is at most as large as $\gamma + \delta c$
z	Any inclusionwise maximal solution has at least z items from group g included into the knapsack

Since all items have a profit-weight ratio close to 1, except for items in the last group for which the profits and weights are very small, we expect the objective function value of any inclusionwise maximal solution to be relatively close to the knapsack capacity c, because it is always possible to nearly fill the knapsack. Lemma 2 formalizes this intuition and demonstrates a lower bound for the objective function value of any inclusionwise maximal solution.

Lemma 2. *Let $\omega \in \Omega(n,c,g,f,\epsilon,s,b)$ be a problem instance, let $x_{\omega,1}, x_{\omega,2}, \ldots, x_{\omega,n}$ be an inclusionwise maximal solution relative to ω and let $d = \min(g-1, \lfloor \log_b(m) \rfloor)$. The objective function value of any inclusionwise maximal solution $x_{\omega,1}, x_{\omega,2}, \ldots, x_{\omega,n}$ is at least as large as $\alpha(c-\beta)$, where $\alpha = \frac{\lfloor(\frac{1}{b^{g-1}}+\epsilon)c\rfloor+1}{\lfloor(\frac{1}{b^{g-1}}+\epsilon)c\rfloor+s}$ and $\beta = \lfloor(\frac{1}{b^d}+\epsilon)c\rfloor + s + s(n-m(g-1))$.*

Proof. Because of Lemma 1, there exists an item j in group d such that $x_{\omega,j}=0$. It follows that $\sum_{i=1}^{n} w_{\omega,i}x_{\omega,i} > c - (\lfloor(\frac{1}{b^d}+\epsilon)c\rfloor + s)$, because otherwise we could include item j and thus $x_{\omega,1}, x_{\omega,2}, \ldots, x_{\omega,n}$ would not be an inclusionwise maximal solution relative to ω.

Let F denote the set of all indices of items that belong to the first $g-1$ groups and let L denote the set of all indices of items that belong to the last group. Since the number of items in L is equal to $n-m(g-1)$, we also have the inequality:

$$\sum_{i \in L} w_{\omega,i}x_{\omega,i} \le s(n-m(g-1))$$

We can now derive a lower bound for the objective function value of $x_{\omega,1}, x_{\omega,2}, \ldots, x_{\omega,n}$:

$$\sum_{i=1}^{n} p_{\omega,i}x_{\omega,i} \ge \sum_{i \in F} p_{\omega,i}x_{\omega,i} = \sum_{i \in F} w_{\omega,i}\frac{p_{\omega,i}}{w_{\omega,i}}x_{\omega,i}$$
$$\ge \min_{i \in F}\left(\frac{p_{\omega,i}}{w_{\omega,i}}\right)\sum_{i \in F} w_{\omega,i}x_{\omega,i}$$

Since

$$\min_{i \in F}\left(\frac{p_{\omega,i}}{w_{\omega,i}}\right) \ge \frac{\left\lfloor\left(\frac{1}{b^{g-1}}+\epsilon\right)c\right\rfloor+1}{\left\lfloor\left(\frac{1}{b^{g-1}}+\epsilon\right)c\right\rfloor+s} = \alpha$$

and

$$\sum_{i \in F} w_{\omega,i}x_{\omega,i} = \sum_{i=1}^{n} w_{\omega,i}x_{\omega,i} - \sum_{i \in L} w_{\omega,i}x_{\omega,i} > c - \left(\left\lfloor\left(\frac{1}{b^d}+\epsilon\right)c\right\rfloor + s\right)$$
$$-\sum_{i \in L} w_{\omega,i}x_{\omega,i} \ge c - \left(\left\lfloor\left(\frac{1}{b^d}+\epsilon\right)c\right\rfloor + s\right) - s(n-m(g-1)) = c-\beta$$

we obtain $\sum_{i=1}^{n} p_{\omega,i}x_{\omega,i} > \alpha(c-\beta)$, as desired. □

Expected relationships between the parameters By choosing the parameters of the problem instance generator such that c is much larger than sb^g, c is much larger than sn, b^d is much larger than 1 and ϵ is approximately 0, we obtain that α is approximately 1 and c is much larger than β. Hence, by choosing the parameters like this, the objective function value of any inclusionwise maximal solution is indeed relatively close to c, as desired.

Lemma 2 gives us an expression that bounds the objective function value of any inclusionwise maximal solution from below. The last lemma that we need before we can prove Theorem 1, will provide an upper bound for the objective function value of the optimal solution for a problem instance. We can again intuitively expect that this upper bound will be relatively close to the knapsack capacity c, because all items have a profit-weight ratio close to 1, except for items with small weights and profits in the last group. This intuition is made more precise in Lemma 3.

Lemma 3. *Let $\omega \in \Omega(n,c,g,f,\epsilon,s,b)$ be a problem instance and let $x^*_{\omega,1}, x^*_{\omega,2}, \ldots, x^*_{\omega,n}$ be an optimal solution for ω. The objective function value of the optimal solution $x^*_{\omega,1}, x^*_{\omega,2}, \ldots, x^*_{\omega,n}$ is at most as large as $\gamma + \delta c$, where $\gamma = s(n-m(g-1))$ and $\delta = \frac{\lfloor(\frac{1}{b^{g-1}}+\epsilon)c\rfloor+s}{\lfloor(\frac{1}{b^{g-1}}+\epsilon)c\rfloor+1}$.*

Proof. Let F denote the set of all indices of items that belong to the first $g-1$ groups and let L denote the set of all indices of items that belong to the last group. We can now derive an upper bound for the objective function value of the optimal solution $x^*_{\omega,1}, x^*_{\omega,2}, \ldots, x^*_{\omega,n}$ by writing it as a sum of two terms and deriving an upper bound for each term:

$$\sum_{i=1}^{n} p_{\omega,i}x^*_{\omega,i} = \sum_{i\in L} p_{\omega,i}x^*_{\omega,i} + \sum_{i\in F} p_{\omega,i}x^*_{\omega,i}$$

For the first term, we have $\sum\limits_{i\in L} p_{\omega,i}x^*_{\omega,i} \le s(n-m(g-1)) = \gamma$ and for the second term we have:

$$\sum_{i\in F} p_{\omega,i}x^*_{\omega,i} = \sum_{i\in F} w_{\omega,i}\frac{p_{\omega,i}}{w_{\omega,i}}x^*_{\omega,i} \le \max_{i\in F}\left(\frac{p_{\omega,i}}{w_{\omega,i}}\right)\sum_{i\in F} w_{\omega,i}x^*_{\omega,i}$$
$$\le \frac{\left\lfloor\left(\frac{1}{b^{g-1}}+\epsilon\right)c\right\rfloor + s}{\left\lfloor\left(\frac{1}{b^{g-1}}+\epsilon\right)c\right\rfloor + 1}\sum_{i\in F} w_{\omega,i}x^*_{\omega,i} \le \frac{\left\lfloor\left(\frac{1}{b^{g-1}}+\epsilon\right)c\right\rfloor + s}{\left\lfloor\left(\frac{1}{b^{g-1}}+\epsilon\right)c\right\rfloor + 1}c = \delta c$$

such that $\sum\limits_{i=1}^{n} p_{\omega,i}x^*_{\omega,i} \le \gamma + \delta c$, as desired. □

Expected relationships between the parameters By choosing the parameters of the problem instance generator such that c is much larger than sb^g and c is much larger than sn, we obtain that c is much larger than γ and δ is approximately 1. Hence, by choosing the parameters like this, the objective function value of an optimal solution is indeed relatively close to c, as desired.

We are now ready to prove the main theorem of this subsection, which states that inclusionwise maximal solutions are indeed near-optimal by showing a lower bound for the worst-case performance ratio between the objective function value of an inclusionwise maximal solution and an optimal solution.

Theorem 1. *Let $\omega \in \Omega(n,c,g,f,\epsilon,s,b)$ be a problem instance, let $x_{\omega,1}, x_{\omega,2}, \ldots, x_{\omega,n}$ be an inclusionwise maximal solution relative to ω and let $x^*_{\omega,1}, x^*_{\omega,2}, \ldots, x^*_{\omega,n}$ be an optimal solution for ω. The ratio between the objective function value of the inclusionwise maximal solution $x_{\omega,1}, x_{\omega,2}, \ldots, x_{\omega,n}$ and the optimal solution $x^*_{\omega,1}, x^*_{\omega,2}, \ldots, x^*_{\omega,n}$ is at least as large as $\frac{\alpha(c-\beta)}{\gamma+\delta c}$, where $\alpha = \frac{\lfloor(\frac{1}{b^{g-1}}+\epsilon)c\rfloor+1}{\lfloor(\frac{1}{b^{g-1}}+\epsilon)c\rfloor+s}$, $\beta = \lfloor(\frac{1}{b^d}+\epsilon)c\rfloor + s + s(n-m(g-1))$, $\gamma = s(n-m(g-1))$ and $\delta = \frac{\lfloor(\frac{1}{b^{g-1}}+\epsilon)c\rfloor+s}{\lfloor(\frac{1}{b^{g-1}}+\epsilon)c\rfloor+1}$.*

Proof. Using Lemmas 2 and 3, we immediately obtain:

$$\min_{\omega\in\Omega(n,c,g,f,\epsilon,s,b)} \frac{\sum_{i=1}^{n} p_{\omega,i}x_{\omega,i}}{\sum_{i=1}^{n} p_{\omega,i}x^*_{\omega,i}} \ge \frac{\min_{\omega\in\Omega(n,c,g,f,\epsilon,s,b)}\sum_{i=1}^{n} p_{\omega,i}x_{\omega,i}}{\max_{\omega\in\Omega(n,c,g,f,\epsilon,s,b)}\sum_{i=1}^{n} p_{\omega,i}x^*_{\omega,i}}$$
$$\ge \frac{\alpha(c-\beta)}{\gamma+\delta c} \qquad \square$$

Expected relationships between the parameters By combining the parameter choices of Lemmas 2 and 3, we obtain that α is approximately 1, c is much larger than β, c is much larger than γ and δ is approximately 1. Hence, by choosing the parameters like this, inclusionwise maximal solutions are indeed near-optimal, as desired.

We finish this subsection by briefly illustrating Lemmas 2, 3 and Theorem 1 with a numerical example. For any problem instance $\omega \in \Omega(10^3, 10^{10}, 11, 10^{-1}, 10^{-5}, 300, 2)$ and any inclusionwise maximal solution $x_{\omega,1}, x_{\omega,2}, \ldots, x_{\omega,n}$ relative to ω, we have $m = 90$, $d = 6$, $\alpha \approx 0.99997$ and $\beta \approx 0.016c$ such that the objective function value of the solution $x_{\omega,1}, x_{\omega,2}, \ldots, x_{\omega,n}$ is at least as large as $\alpha(c-\beta) \approx 0.984c$. For any optimal solution $x^*_{\omega,1}, x^*_{\omega,2}, \ldots, x^*_{\omega,n}$, we have $\gamma \approx 0.000003c$ and $\delta \approx 1.00003$ such that the objective function value of the optimal solution $x^*_{\omega,1}, x^*_{\omega,2}, \ldots, x^*_{\omega,n}$ is at most as large as $\gamma + \delta c \approx 1.00003c$. Hence, the worst-case ratio between the objective function value of an inclusionwise maximal solution and an optimal solution for any problem instance $\omega \in \Omega(10^3, 10^{10}, 11, 10^{-1}, 10^{-5}, 300, 2)$ is at least as large as $\frac{\alpha(c-\beta)}{\gamma+\delta c} \approx 0.98397$. This means that the optimality gap is at most as large as $1 - 0.98397 \approx 1.603\%$ and so $x_{\omega,1}, x_{\omega,2}, \ldots, x_{\omega,n}$ is near-optimal.

5.2. Varying profit-weight ratios in optimal solutions

Recall from Section 3 that most successful algorithms for solving the 0–1 knapsack problem rely on so-called *cores*. These algorithms use the observation that for many typical problem instances the structure of an optimal solution is often very similar to the structure of a certain greedy solution, except for items in the core (Martello & Toth, 1988; Pisinger, 1995; 1997). The greedy algorithm first sorts the items in decreasing order of their profit-weight ratio (i.e. such that $\frac{p_i}{w_i} \ge \frac{p_{i+1}}{w_{i+1}}, \forall i \in \{1, 2, \ldots, n-1\}$). The algorithm then goes over the items one by one in this order and greedily adds an item to the knapsack if this does not violate the capacity constraint. This greedy solution is also used to find the optimal solution of the linear relaxation of the 0–1 knapsack problem (in which fractions of an item can be added to the knapsack). The core then consists of all items for which its rank is between the minimum and maximum rank of an item for which x_i is different in the optimal solution for the 0–1 knapsack problem and the optimal solution for its linear relaxation. This definition of the core concept implies that the optimal solution should be known in order to determine the core. Algorithms like ***Expknap***, ***Minknap*** and ***Combo*** deal with this problem by using expanding cores (see Section 3). These algorithms rely on forward-backward dynamic programming starting from an item set that only contains the break item and this set is gradually expanded. When the algorithm terminates, it will have enumerated a core.

In case the core is really big, such algorithms might become slow. In order to construct problem instances with big cores, one could try to enforce items to have a wide variety of profit-weight ratios. However, this will often just result in the items with low profit-weight ratio not being part of the optimal solution. Thus, great care must be taken to construct such problem instances. For the NMGE problem instances from this paper, the items from the first $g-1$ groups all have a profit-weight ratio close to 1, whereas the items from the last group have a wide variety of profit-weight ratios. By carefully choosing the parameters of the problem instance generator, we can guarantee that any optimal solution must contain at least z items from the last group of items, for any integer z between 0 and the size of the last group. This is precisely due to the (somewhat complicated) way in which the weights of the items are defined. This statement is formalized and proven in the following theorem.

Theorem 2. *Let $\omega \in \Omega(n,c,g,f,\epsilon,s,b)$ be a problem instance. If z is an integer between 0 and $n-(g-1)m$ (the size of the last group) such that $\frac{c}{b^{g-1}} - (g-1)m(\epsilon c + s) \ge sz$, then any inclusionwise maximal solution $x_{\omega,1}, x_{\omega,2}, \ldots, x_{\omega,n}$ relative to ω contains at least z items from the last group.*

Proof. The weight of item j in group i ($1 \le i \le g-1$) is defined as $w_{\omega,j} = \lfloor(\frac{1}{b^i}+\epsilon)c\rfloor + r_{2,j}$, and thus we have the inequality:

$$\left(\frac{1}{b^i}+\epsilon\right)c < w_{\omega,j} \le \left(\frac{1}{b^i}+\epsilon\right)c + s$$

We can slightly rewrite this to obtain a form that will be more useful in the next paragraph:

$$\frac{b^{g-1-i}}{b^{g-1}}c + \epsilon c < w_{\omega,j} \le \frac{b^{g-1-i}}{b^{g-1}}c + \epsilon c + s$$

Let F denote the set of all indices of items that belong to the first $g-1$ groups. If we select l items from F, there exists some non-negative integer k such that:

$$\frac{k}{b^{g-1}}c + l\epsilon c < \sum_{a\in F} w_{\omega,a}x_{\omega,a} \leq \frac{k}{b^{g-1}}c + l(\epsilon c + s)$$

The above expression is obtained by taking the sum of l inequalities corresponding to the l selected items from F. For the inclusionwise maximal solution $x_{\omega,1}, x_{\omega,2}, \ldots, x_{\omega,n}$, it must be true that $k \leq b^{g-1} - 1$, because otherwise the capacity constraint would be violated since for $k \geq b^{g-1}$ we have:

$$\sum_{a=1}^{n} w_{\omega,a}x_{\omega,a} \geq \sum_{a\in F} w_{\omega,a}x_{\omega,a} > \frac{b^{g-1}}{b^{g-1}}c + l\epsilon c \geq c$$

Since $k \leq b^{g-1} - 1$ and $l \leq (g-1)m$ (the number of items in F), we also obtain:

$$c - \sum_{a\in F} w_{\omega,a}x_{\omega,a} \geq c - \left(\frac{b^{g-1}-1}{b^{g-1}}c + (g-1)m(\epsilon c + s)\right)$$
$$= \frac{c}{b^{g-1}} - (g-1)m(\epsilon c + s) \geq sz$$

Because of this inequality and the fact that $x_{\omega,1}, x_{\omega,2}, \ldots, x_{\omega,n}$ is an inclusionwise maximal solution, it must contain at least z items from the last group. □

Although problem instances with large cores tend to be harder to solve, great care must be taken in interpreting Theorem 2. One might be tempted to think that choosing the parameters of the problem instance generator such that z is large, will automatically lead to hard problem instances. However, this is not necessarily true, because in this case Theorem 2 also reveals a part of the structure of the optimal solution and the problem instance can be reduced to a smaller problem instance in some cases. This potential pitfall will also come back later in the experiments section. Hence, Theorem 2 gives additional insight into the structure of the optimal solution and problem instances for which this structure is revealed tend to be easier.

6. Experiments

We have systematically generated a large set of hard NMGE problem instances using the problem instance generator described in this paper. A full factorial design of experiments was used, where we have generated a problem instance for every combination of parameters with $n \in \{400, 600, 800, 1000, 1200\}$, $c \in \{10^6, 10^8, 10^{10}\}$, $g \in \{2, 6, 10, 14\}$, $f \in \{0.1, 0.2, 0.3\}$, $\epsilon \in \{0, 10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}\}$, $s \in \{100, 200, 300\}$ and $b \in \{2\}$. The parameters were chosen such that (for most combinations) the assumptions explained in Section 5 are met. For the integer parameter b, we made the decision to always set $b = 2$ in the experiments for several reasons. This parameter occurs as the base of an exponent and hence it is very sensitive. If we consider problem instances where $b \geq 3$, there are many problem instances for which the assumptions mentioned in Section 5 are not met since for example $c < sb^g$ for many combinations. As a result, choosing $b \geq 3$ would lead to many degenerate problem instances in which not all groups are pairwise disjoint (i.e. the profits and weights of items in different groups are almost equal to each other) and this only complicates the interpretation of the results. Finally, choosing $b = 2$ also results in a dataset which is more manageable and this also allows us to generate more combinations for the other parameters. The resulting dataset contains $5 \times 3 \times 4 \times 3 \times 6 \times 3 \times 1 = 3240$ problem instances. The experiments were all conducted on the ThinKing cluster of the Flemish Supercomputer Center (VSC), using 10 GB RAM memory for each run and powerful CPUs with a clock rate of 2.5 gigahertz. Completing all the experiments on this hardware took around 810 CPU-hours.

6.1. Preliminary experiment

It is hard to make general statements about the hardness of a problem instance with regard to all possible algorithms. Hence, the hardness of a problem instance is often characterized as empirical hardness by measuring the runtime of a certain algorithm. It is often the case that one algorithm performs better on problem instances with one specific structure whereas other algorithms perform better on problem instances with another structure. For the 0–1 knapsack problem, however, it is widely known that the ***Combo*** algorithm (Martello et al., 1999) is superior to all other available algorithms that have been developed so far on a large set of problem instances. This has already been empirically demonstrated before by Martello et al. (1999) and also by Pisinger (2005) for a large set of problem instances consisting of 31,800 problem instances from 13 different classes. To verify that this observation also holds for our instances, we have performed a preliminary control experiment in which we compared ***Combo***[2] with two other algorithms, namely ***Expknap***[3] (Pisinger, 1995) and ***Minknap***[2,3] (Pisinger, 1997). The code for these three algorithms has been previously made available at http://hjemmesider.diku.dk/~pisinger/codes.html.

Since the problem instances that we propose are very hard, it would be computationally infeasible to run these three algorithms on all 3240 problem instances. Hence, we have decided for this preliminary experiment to run these three algorithms on 100 randomly selected problem instances from the whole dataset, using a time limit of 7200 seconds per run. From this preliminary experiment it was indeed reconfirmed that ***Combo*** was a lot faster than ***Expknap*** and ***Minknap***. More specifically, there were only 6 problem instances out of 100 which could not be solved by ***Combo*** within the time limit, whereas there were 79 and 34 such instances for ***Expknap*** and ***Minknap*** respectively. Hence, in the rest of this paper we will use the runtime of ***Combo*** to empirically measure the hardness of the problem instances.

6.2. Comparison with other problem instances

6.2.1. Comparison with problem instances from Pisinger (2005)

We compared the NMGE problem instances from this paper with the problem instances from Pisinger (2005). One of the main goals of Pisinger (2005) was to empirically demonstrate that there exist problem instances with a certain structure that are hard to solve for exact algorithms for the 0–1 knapsack problem, despite the fact that many large problem instances can be solved in several (milli)seconds. This goal is aligned with one of the main goals of this paper. Pisinger's dataset is well known and has been widely used by various other researchers. It consists of 13 different classes of problem instances which collectively yield 31,800 problem instances. These problem instances, their solutions and the runtimes by ***Combo*** have previously been made available at http://hjemmesider.diku.dk/~pisinger/codes.html.

To be able to make a fair comparison with our instances, we have rerun the 3000 most difficult problem instances from Pisinger (2005) on the same hardware as our instances using a time limit of 7200 seconds per run. The evaluation metric that we use to com-

[2] The variable *MAXSTATES* was increased to 450,000,000 to avoid running out of memory.

[3] The code was slightly changed to be able to handle 64-bit integers instead of 32-bit integers.

Table 2
Summary of the boxplots from Figs. 1–3 and 7–12. All times in this table are expressed in seconds.

Boxplot	Left whisker	First quartile	Second quartile	Third quartile	Right whisker	Average	Timeouts
Fig. 1: *this paper*	**3.45×10^{-2}**	**1.74×10^{-1}**	**8.73×10^{-1}**	**2.03×10^{1}**	**5.02×10^{1}**	**1.24×10^{3}**	**263**
Fig. 1: *Pisinger3000*	1.38×10^{-2}	7.96×10^{-2}	2.38×10^{-1}	1.09×10^{0}	2.59×10^{0}	1.15×10^{0}	0
Fig. 2: *this paper*, $n \leq 500$	3.80×10^{-2}	1.30×10^{-1}	3.91×10^{-1}	5.97×10^{0}	1.47×10^{1}	6.92×10^{2}	30
Fig. 2: *Pisinger3000*, $n \leq 500$	1.38×10^{-2}	1.76×10^{-1}	5.88×10^{-1}	1.86×10^{0}	4.36×10^{0}	1.32×10^{0}	0
Fig. 2: *this paper*, $501 \leq n \leq 1000$	3.45×10^{-2}	1.74×10^{-1}	9.90×10^{-1}	2.19×10^{1}	5.43×10^{1}	1.31×10^{3}	**167**
Fig. 2: *Pisinger3000*, $501 \leq n \leq 1000$	2.56×10^{-2}	2.00×10^{-1}	1.05×10^{0}	2.62×10^{0}	5.71×10^{0}	2.34×10^{0}	0
Fig. 2: *this paper*, $n > 1000$	**4.30×10^{-2}**	**2.55×10^{-1}**	**1.49×10^{0}**	**4.63×10^{1}**	**1.14×10^{2}**	**1.55×10^{3}**	66
Fig. 2: *Pisinger3000*, $n > 1000$	1.66×10^{-2}	5.04×10^{-2}	1.10×10^{-1}	3.99×10^{-1}	9.20×10^{-1}	8.37×10^{-1}	0
Fig. 3: *initial population*	1.02×10^{-2}	1.49×10^{-2}	2.73×10^{-2}	4.45×10^{-2}	8.90×10^{-2}	3.59×10^{-2}	**0**
Fig. 3: *evolved instances*	9.45×10^{-3}	1.27×10^{-2}	2.26×10^{-2}	3.19×10^{-2}	6.03×10^{-2}	2.59×10^{-2}	**0**
Fig. 3: *this paper small instances*	**3.45×10^{-2}**	**9.93×10^{-2}**	**1.73×10^{-1}**	**3.22×10^{-1}**	**6.32×10^{-1}**	**2.99×10^{-1}**	**0**
Fig. 7: $n = 400$	3.80×10^{-2}	1.30×10^{-1}	3.91×10^{-1}	5.97×10^{0}	1.47×10^{1}	6.92×10^{2}	30
Fig. 7: $n = 600$	3.76×10^{-2}	1.45×10^{-1}	7.72×10^{-1}	1.14×10^{1}	2.81×10^{1}	1.14×10^{3}	49
Fig. 7: $n = 800$	3.81×10^{-2}	1.72×10^{-1}	1.00×10^{0}	2.26×10^{1}	5.36×10^{1}	1.31×10^{3}	55
Fig. 7: $n = 1000$	3.45×10^{-2}	2.13×10^{-1}	1.37×10^{0}	3.33×10^{1}	7.91×10^{1}	1.49×10^{3}	63
Fig. 7: $n = 1200$	**4.30×10^{-2}**	**2.55×10^{-1}**	**1.49×10^{0}**	**4.63×10^{1}**	**1.14×10^{2}**	**1.55×10^{3}**	**66**
Fig. 8: $c = 10^{6}$	3.45×10^{-2}	1.02×10^{-1}	1.78×10^{-1}	4.00×10^{-1}	8.33×10^{-1}	4.31×10^{-1}	0
Fig. 8: $c = 10^{8}$	**6.35×10^{-2}**	3.45×10^{-1}	3.11×10^{0}	1.76×10^{1}	4.25×10^{1}	2.57×10^{1}	0
Fig. 8: $c = 10^{10}$	3.80×10^{-2}	**5.94×10^{-1}**	**3.22×10^{1}**	**4.23×10^{3}**	**6.47×10^{3}**	**3.68×10^{3}**	**263**
Fig. 9: $g = 2$	3.80×10^{-2}	1.19×10^{-1}	1.82×10^{-1}	3.21×10^{-1}	6.22×10^{-1}	2.57×10^{-1}	0
Fig. 9: $g = 6$	**4.34×10^{-2}**	**8.21×10^{-1}**	2.27×10^{0}	8.97×10^{0}	2.07×10^{1}	9.49×10^{0}	0
Fig. 9: $g = 10$	3.81×10^{-2}	1.82×10^{-1}	**1.77×10^{1}**	1.95×10^{2}	4.86×10^{2}	1.47×10^{3}	71
Fig. 9: $g = 14$	3.45×10^{-2}	1.80×10^{-1}	3.66×10^{0}	**1.00×10^{3}**	**1.85×10^{3}**	**3.47×10^{3}**	**192**
Fig. 10: $\epsilon = 0$	4.67×10^{-2}	1.88×10^{-1}	7.25×10^{-1}	2.36×10^{1}	5.78×10^{1}	9.07×10^{2}	33
Fig. 10: $\epsilon = 10^{-5}$	3.81×10^{-2}	2.01×10^{-1}	**1.47×10^{0}**	**5.69×10^{1}**	**1.41×10^{2}**	**1.99×10^{3}**	**73**
Fig. 10: $\epsilon = 10^{-4}$	3.45×10^{-2}	1.67×10^{-1}	1.03×10^{0}	2.03×10^{1}	4.42×10^{1}	1.49×10^{3}	51
Fig. 10: $\epsilon = 10^{-3}$	3.76×10^{-2}	1.40×10^{-1}	4.80×10^{-1}	1.87×10^{1}	4.41×10^{1}	1.69×10^{3}	61
Fig. 10: $\epsilon = 10^{-2}$	4.49×10^{-2}	1.63×10^{-1}	8.92×10^{-1}	2.93×10^{1}	7.25×10^{1}	1.28×10^{3}	45
Fig. 10: $\epsilon = 10^{-1}$	**5.21×10^{-2}**	**2.09×10^{-1}**	7.84×10^{-1}	1.26×10^{1}	2.97×10^{1}	5.89×10^{1}	0
Fig. 11: $f = 0.1$	3.45×10^{-2}	1.42×10^{-1}	7.56×10^{-1}	1.40×10^{1}	3.41×10^{1}	**1.27×10^{3}**	**92**
Fig. 11: $f = 0.2$	**4.34×10^{-2}**	1.80×10^{-1}	8.53×10^{-1}	2.18×10^{1}	5.37×10^{1}	1.18×10^{3}	83
Fig. 11: $f = 0.3$	3.81×10^{-2}	**2.16×10^{-1}**	**1.05×10^{0}**	**3.16×10^{1}**	**7.58×10^{1}**	1.25×10^{3}	88
Fig. 12: $s = 100$	**4.51×10^{-2}**	**1.76×10^{-1}**	**1.00×10^{0}**	**2.76×10^{1}**	**6.84×10^{1}**	1.21×10^{3}	86
Fig. 12: $s = 200$	3.80×10^{-2}	1.70×10^{-1}	8.76×10^{-1}	2.09×10^{1}	4.99×10^{1}	**1.28×10^{3}**	**91**
Fig. 12: $s = 300$	3.45×10^{-2}	1.72×10^{-1}	7.31×10^{-1}	1.66×10^{1}	4.10×10^{1}	1.22×10^{3}	86

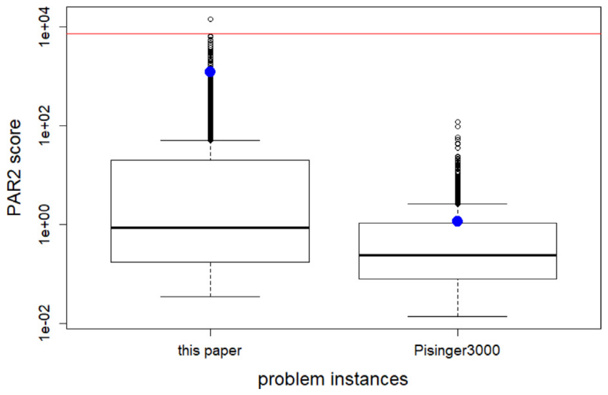


Fig. 1. PAR2 scores for different problem instances.

pare the hardness of the problem instances is the PAR2 score.[4] The PAR2 score is equal to the runtime of an algorithm, but penalizes algorithm runs that do not produce the correct solution within the time limit by assigning a score of $2 \times 7200 = 14,400$ seconds. The PAR2 scores of the 3000 most difficult problem instances from Pisinger (2005) (*Pisinger3000*) and the problem instances from this paper are shown in the boxplots in Fig. 1. We also indicated the average runtimes (thick dots) and the time limit (horizontal line).

These boxplots show that the instances that we propose in this paper are several orders of magnitude harder than the hardest instances that were previously known. Note that the results are shown on a logarithmic scale. Although the median PAR2 scores for both datasets are still relatively low (8.73×10^{-1} seconds for the instances in this paper versus 2.38×10^{-1} seconds for *Pisinger3000*), it is more interesting to look at the hardest problem instances on the right side of the third quartile. The third quartile is 2.03×10^{1} seconds for the instances in this paper versus 1.09×10^{0} seconds for *Pisinger3000*. There were 263 out of 3240 problem instances from our dataset which could not be solved by ***Combo*** within the time limit of 7200 seconds, whereas there were 0 out of 3000 such problem instances for *Pisinger3000*. The hardest problem instance from *Pisinger3000* only took 1.21×10^{2} seconds to solve versus more than 7200 seconds for the instances in this paper, which yields a ratio of around 60. The average PAR2 scores are quite different as well: 1.24×10^{3} seconds for the instances in this paper versus 1.15×10^{0} seconds for *Pisinger3000*. These numbers are also summarized in Table 2. Hence, these numbers indicate that the NMGE problem instances from this paper are harder than the problem instances from *Pisinger3000*.

It is also remarkable that the problem instances from this paper are relatively small in comparison with *Pisinger3000* (see Table 3). The problem variables n and c have a direct impact on the size of the search space (e.g. ***Combo*** has a worst-case time complexity of $O(nc)$ Martello et al., 1999). The average values for n and c for the problem instances in *Pisinger3000* are both approximately five times larger than for this paper and the ratio between the average

[4] We have chosen PAR2 instead of the more widely used variant PAR10 to decrease the effect of runs that do not produce the result within the time limit.

Table 3
Comparison of the number of items n, the knapsack capacity c and the average PAR2 score for the problem instances from this paper and Pisinger (2005).

Problem instances	Average n	Average c	Average PAR2 score
this paper	800.0	3.37×10^9	1.24×10^3 seconds
Pisinger3000	4022.0	1.52×10^{10}	1.15×10^0 seconds

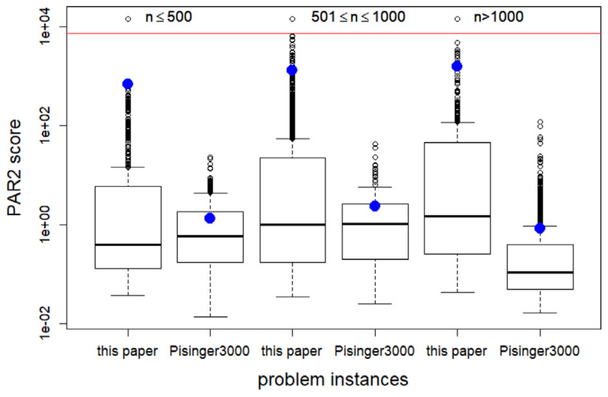


Fig. 2. PAR2 scores for different problem instances where n varies.

Table 4
Comparison of the number of items n, the knapsack capacity c and the average PAR2 score for the initial population from Smith-Miles et al. (2021), the evolved problem instances from Smith-Miles et al. (2021) and small problem instances from this paper.

Problem instances	Average n	Average c	Average PAR2 score
initial population	1000.0	2.61×10^5	3.59×10^{-2} seconds
evolved instances	1000.0	3.09×10^5	2.59×10^{-2} seconds
this paper small instances	1000.0	1.00×10^6	2.99×10^{-1} seconds

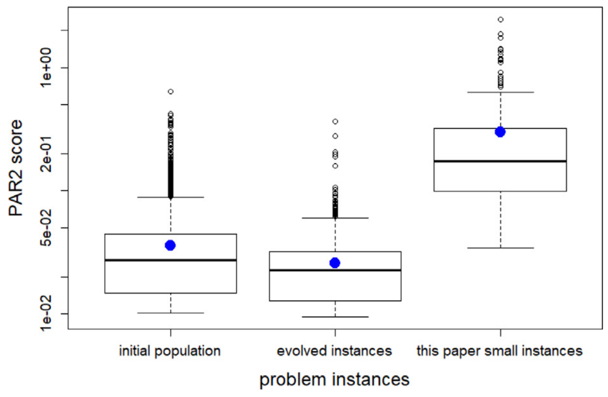


Fig. 3. PAR2 scores for small problem instances.

PAR2 score is approximately 1078.26. Hence, we can conclude that our problem instances are harder to solve, despite much smaller search spaces.

We also grouped the problem instances from this paper and *Pisinger3000* into three coarse bins based on the number of items n. For the problem instances in the first, second and third bin we respectively have $n \leq 500$, $501 \leq n \leq 1000$ and $n > 1000$. The boxplots for each of these can be found in Fig. 2. This figure shows that the conclusions that could be drawn when comparing both datasets as a whole (Fig. 1) can also be drawn when comparing both datasets for every bin individually. The ratios between the average PAR2 scores for both datasets are respectively 524.24, 559.83 and 1851.85 for the first, second and third bin. This indicates for every bin that the problem instances from this paper are harder than the problem instances from *Pisinger3000* and the ratios between the average PAR2 scores increase for increasing values of n.

6.2.2. Comparison with problem instances from Smith-Miles et al. (2021)

Smith-Miles et al. (2021) recently proposed problem instances that cover even more parts of the problem instance space than the instances from Pisinger (2005). To generate these new problem instances, they first identify a set of 26 target points that represent combinations of features of problem instances that are not present in current datasets. For every target point, they then use two genetic algorithms (for generating either weakly or strongly structured problem instances) that evolve an initial population of problem instances towards the target point. The initial population consists of problem instances, each having $n = 1000$ items, from the classes proposed by Pisinger (2005) as well as several newly introduced classes by Smith-Miles et al. (2021) and these collectively yield 4000 problem instances, whereas the evolved problem instances collectively yield 1300 problem instances.

An important remark is that the problem instances proposed by Smith-Miles et al. (2021) have much smaller knapsack capacities than those proposed by Pisinger and this leads to much smaller runtimes than those from Pisinger (2005) and this paper (see Table 4). All problem instances from Smith-Miles et al. (2021) can be solved within 1 second by ***Combo***. Hence, the comparison in this subsection was mainly carried out for the sake of completeness and we will particularly focus on (i) whether the evolved instances are harder than those in the initial population, acknowledging that they were designed to be diverse rather than hard, (ii) how these problem instances compare to the problem instances with the smallest knapsack capacity ($c = 10^6$) and the same number of items ($n = 1000$) from this paper and (iii) whether the instances from the current paper further increase the diversity of the problem instances with respect to several features.

The PAR2 scores of the problem instances in the initial population from Smith-Miles et al. (2021), the evolved problem instances from Smith-Miles et al. (2021) and the small problem instances from the current paper can be found in Fig. 3. The three quartiles and the average PAR2 score of the evolved problem instances are all slightly lower than those of the initial population (average PAR2 score of 3.59×10^{-2} seconds for the initial population versus 2.59×10^{-2} seconds for the evolved problem instances). The PAR2 scores of the hardest evolved problem instances are also slightly lower than those of the initial population. Hence, these numbers all lead to the conclusion that the evolved problem instances are not harder than the problem instances in the initial population, although they are more diverse. The small problem instances from the current paper have knapsack capacities that are on average 3.24 times larger than the evolved problem instances from Smith-Miles et al. (2021) and have an average PAR2 score that is 11.54 times as large. The ratios between the three quartiles are similar (they are 7.81, 7.65 and 10.09 for respectively the first, second and third quartile). This suggests that the small problem instances from the current paper are also harder than those from Smith-Miles et al. (2021) although they also have larger knapsack capacities and the relative difference is not as large as in the case of the larger problem instances from the previous subsection (with an average PAR2 score of 1.24×10^3 seconds and an average knapsack capacity of $c = 3.37 \times 10^9$).

Since the problem instances from Smith-Miles et al. (2021) were designed to be diverse, it is also interesting to investigate whether the problem instances from the current paper further increase the diversity with respect to several features. To answer this question, we used the Instance Space Analysis methodology developed by Smith-Miles, Baatar, Wreford, & Lewis (2014); Smith-Miles & Bowly (2015); Smith-Miles & Lopes (2012).

This methodology allows us to visualize the problem instances in a two-dimensional space in which patterns regarding algorithm performance and problem instance features can be visually observed. In order to obtain a two-dimensional representation of a problem instance, a number of numerical features is calculated for each problem instance such that a problem instance can be regarded as a point in a high-dimensional feature space. These points in a high-dimensional feature space can then be projected to a two-dimensional space, by first preprocessing the features (e.g. scaling and normalization) and then finding two appropriate linear combinations of the features (one for each dimension).

In this experiment we used the online tool MATILDA (Smith-Miles, 2019), which implements the Instance Space Analysis methodology. To find a projection from a high-dimensional feature space to a two-dimensional space, MATILDA uses the PILOT method (Muñoz, Villanova, Baatar, & Smith-Miles, 2018). According to Smith-Miles et al. (2021), this method finds a projection "such that algorithm performance and feature values increase linearly from one edge of the instance space to the opposite, thereby assisting the visualisation of directions of hardness and feature correlation to support insights". For this experiment, we used 10 different numerical features from Smith-Miles et al. (2021). These features were calculated for each problem instance from this paper and from Smith-Miles et al. (2021).

We obtained the following projection matrix through MATILDA for projecting a problem instance as a point in a 10D (preprocessed) feature space to a 2D space:

$$\begin{bmatrix} Z_1 \\ Z_2 \end{bmatrix} = \begin{bmatrix} -0.2453 & 0.6246 \\ 0.5633 & -0.4492 \\ 0.1321 & 0.4560 \\ -0.6168 & -0.0480 \\ -0.3160 & -0.0847 \\ -0.2619 & 0.1622 \\ 0.5660 & -0.3642 \\ 0.2619 & -0.0222 \\ 0.4804 & 0.4630 \\ 0.4333 & 0.3491 \end{bmatrix}^T \begin{bmatrix} \text{Dominant Pairs} \\ \text{Correlation Coeff.} \\ \text{Approximation Gap} \\ \text{Possible Fix Prop.} \\ \text{First Weight} \\ \text{First Profit} \\ \text{Polyfit Quadratic} \\ \text{Even-Odd Likeness} \\ \text{Greedy Unused Capacity} \\ \text{Red. Coeff. Var. Effic.} \end{bmatrix}$$

These features all have the domain [0,1], except for the feature Correlation Coefficient, which has the domain $[-1, 1]$, and the feature Even-Odd Likeness, which has the domain [0,10]. A precise description of these features can be found in Table 2 from Smith-Miles et al. (2021). The projection gives rise to the 2D instance space shown in Fig. 4 for the problem instances from this paper (blue), the initial population from Smith-Miles et al. (2021) (red) and the evolved instances from Smith-Miles et al. (2021) (green). From this figure we can see that the evolved instances from Smith-Miles et al. (2021) indeed add diversity to the initial population, as intended by Smith-Miles et al. (2021). The problem instances from the current paper are situated in a smaller area, but further increase the diversity of these two sets of problem instances by filling two previously unfilled gaps outside of the previous boundary. The first filled gap is located near $(Z_1, Z_2) = (2.5, 2)$ and the second one is located near $(Z_1, Z_2) = (2.5, 0)$.

The Instance Space Analysis methodology also allows us to visually observe the hard regions of the problem instance space. The PAR2 scores obtained by ***Combo*** are shown using colorbars in Fig. 5 (yellow colors correspond to higher PAR2 scores and blue colors correspond to lower PAR2 scores). This figure visually confirms the conclusions that we made earlier: ***Combo*** is able to solve most problem instances quite fast, except for the problem instances located roughly near $(Z_1, Z_2) = (2.5, 0)$. It is interesting to see that the problem instances from the current paper around the first gap (near $(Z_1, Z_2) = (2.5, 2)$) increase the diversity of the available problem instances, but do not necessarily lead to (much) harder

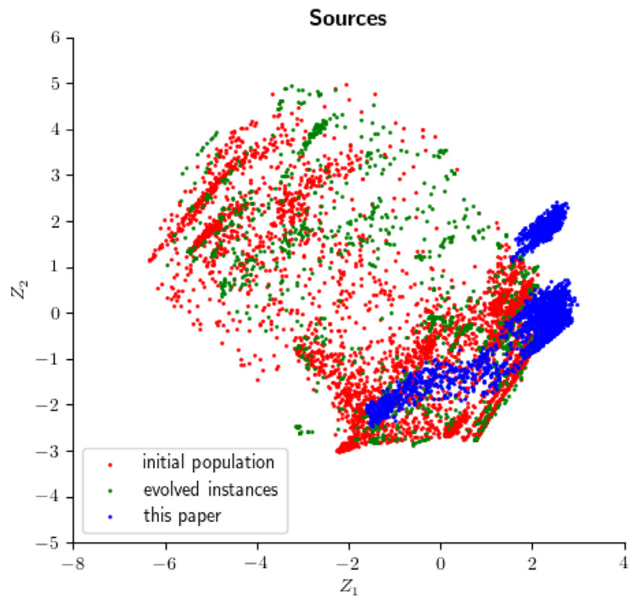


Fig. 4. 2D representation of the problem instances from this paper and Smith-Miles et al. (2021) obtained using MATILDA (Smith-Miles, 2019).

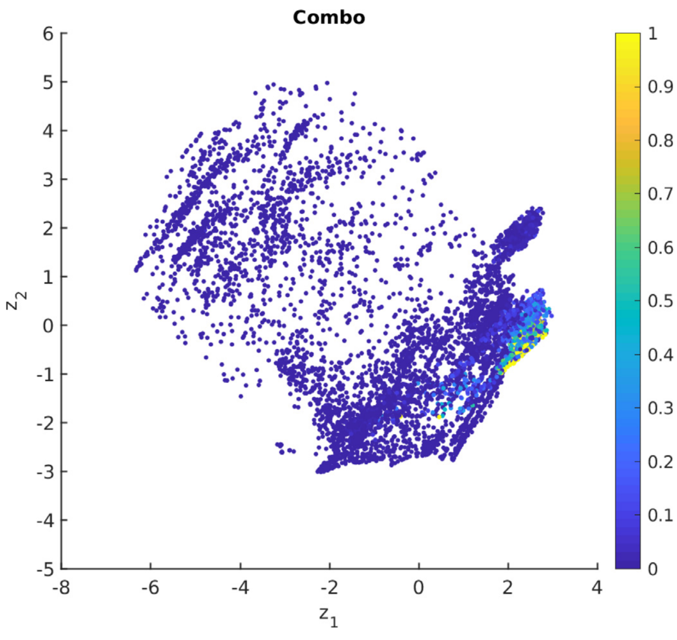
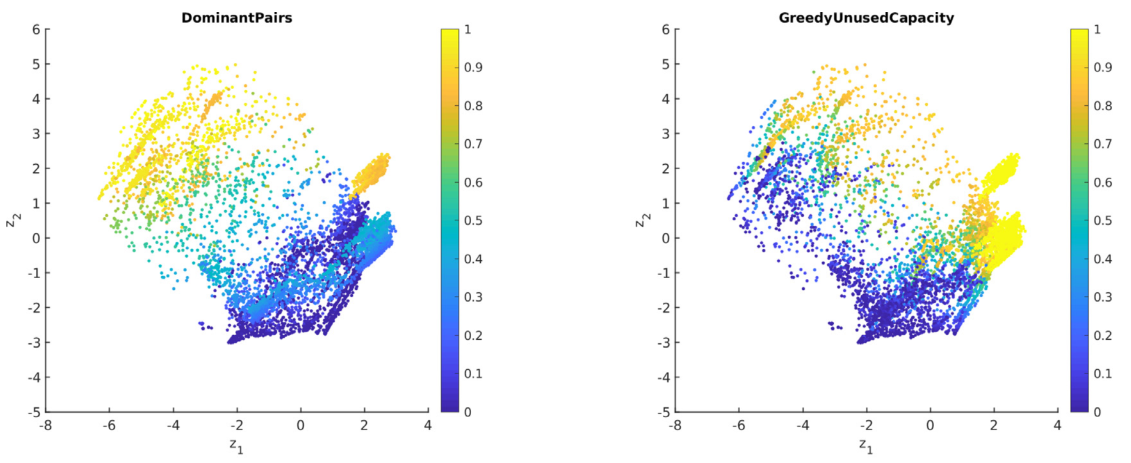


Fig. 5. Visualization of the PAR2 scores obtained by ***Combo***.

problem instances, whereas the problem instances around the second gap (near $(Z_1, Z_2) = (2.5, 0)$) are much harder.

Finally, we also show for two specific features (Dominant Pairs and Greedy Unused Capacity) how they are distributed across the instance space (see Fig. 6). Smith-Miles et al. (2021) describe the Dominant Pairs feature as the "proportion of item pairs i, j for which $p_i \geq p_j$ and $w_i \leq w_j$ OR $p_j \geq p_i$ and $w_j \leq w_i$, excluding identical items" and the Greedy Unused Capacity feature as follows: "If the instance has no item pairs i, j such that $w_i \neq w_j$ then this feature defaults to 0. Otherwise it is defined as the capacity unused by the greedy solution, divided by the smallest non-zero difference between any two items' weights, then normalized using Eq. (2)."

$$\text{normalized feature} \leftarrow \frac{\tan^{-1}(\text{raw feature value}/100)}{\pi/2} \tag{2}$$



(a) Distribution of the Dominant Pairs feature **(b)** Distribution of the Greedy Unused Capacity feature

Fig. 6. Distribution of two features.

This normalization step ensures that different features have comparable magnitudes. The raw features (with domain $\mathbb{R}_{\geq 0}$) are transformed into normalized features (with domain [0,1]).

These two features have a remarkable pattern near the areas where the NMGE problem instances from this paper are located. The Dominant Pairs feature seems to linearly decrease from the upper left corner of the instance space to the lower right corner, except for the areas where the problem instances from this paper are located. Our problem instances near the first gap ($(Z_1, Z_2) = (2.5, 2)$) tend to have a high number of dominated pairs of items and this suggests that such problem instances are easier to solve, since a dominated item can only be included in the optimal knapsack packing if all of the items by which that item is dominated are also included. The Dominant Pairs feature is much lower near the second gap ($(Z_1, Z_2) = (2.5, 0)$) and these problem instances are also much harder. The Greedy Unused Capacity feature is very high for most of our problem instances and reaches higher values than the problem instances from Smith-Miles et al. (2021). In the most difficult area near the second gap this feature is always very high, suggesting that this might be an important feature that correlates with instance hardness, although this feature alone is not sufficient to predict problem instance hardness.

6.3. Influence of parameters on runtime

In this last subsection, we will investigate the influence of the various parameters of the problem instance generator on the time that is needed to solve the generated problem instance. For every parameter, we partitioned the whole dataset (3240 problem instances) according to the different values which that parameter can take. The boxplots of the PAR2 scores for every separate parameter can be found in Figs. 7–12. As before, we indicated the average runtimes by thick dots and the time limit of 7200 seconds by a horizontal line. These figures contain quite a bit of information (i.e. six different numbers for each boxplot, describing the two whiskers, the three quartiles and the average). In the rest of this subsection, we will primarily focus on the general trends that we can see and only to a lesser extent on the precise numbers. For the sake of completeness, we have included the precise numbers that can be derived from the different boxplots in Table 2. For every figure, the maximum values of every column are marked in bold (higher values indicate harder problem instances). Since we are interested in the hard problem instances, amongst these numbers most of our attention will go to the numbers in the rightmost columns since the harder problem instances influence these columns more. This is the case, because the boxplots in the figures indicate strongly right skewed distributions (note that the boxplots are shown on a logarithmic scale).

This subsection can also be placed in the broader context of other studies that deal with so-called *phase transitions*. These phase transitions were originally introduced in statistical mechanics, where they play an important role for concepts such as superfluidity and superconductivity, but they have also been actively studied in the context of combinatorial (optimization) problems under a slightly different meaning (see e.g. Hartmann & Weigt, 2006). In the context of combinatorial optimization, it has been conjectured that for several NP-complete problems there exists at least one parameter and one critical value for that parameter around which the difficulty of the problem instances drastically changes (Achlioptas, Naor, & Peres, 2005; Cheeseman, Kanefsky, & Taylor, 1991). Such phase transitions have for example been observed for the SAT problem (Mitchell, Selman, & Levesque, 1992) and the travelling salesman problem (Gent & Walsh, 1996; Smith-Miles, van Hemert, & Lim, 2010). The parameters that will be discussed in this subsection can also be understood as phase transition parameters as we can also observe clear changes in the difficulty of the problem instances, although these changes are less drastic for some parameters than what has been observed for other problems.

6.3.1. Influence of n and c

Since the parameters n and c have a direct impact on the size of the search space, we can expect that increasing these parameters will cause the PAR2 score to increase as well. This is indeed confirmed by our experiments as can be seen in Figs. 7 and 8. By increasing n and c, the three quartiles of the PAR2 scores also increase. The average PAR2 scores follow the same pattern. It is remarkable that even for the smallest problem instances, with $n = 400$, not all problem instances could be solved within 7200 seconds. More specifically, there were 30 such problem instance out of 648 amongst the group with $n = 400$. For the smallest values of c (10^6 and 10^8), all problem instances could be solved within the time limit. For the largest value of $c = 10^{10}$, however, there were 263 problem instances out of 1080 which could not be solved within 7200 seconds. Hence, the average PAR2 score for this last group (3.68×10^3 seconds) is not very far off the time limit. These summarizing statistics all indicate that the PAR2 scores indeed increase with increasing n and c, which is consistent with what could be intuitively expected.

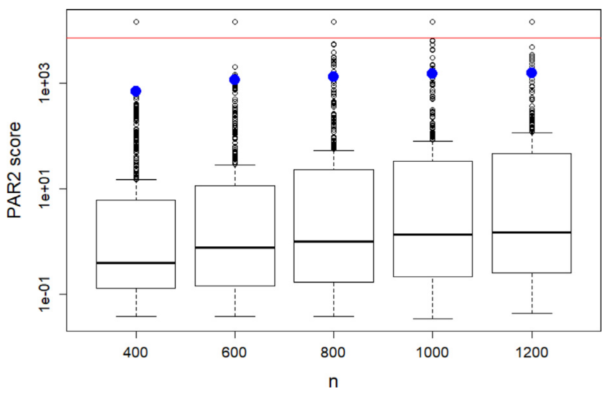


Fig. 7. PAR2 scores for different values of *n*.

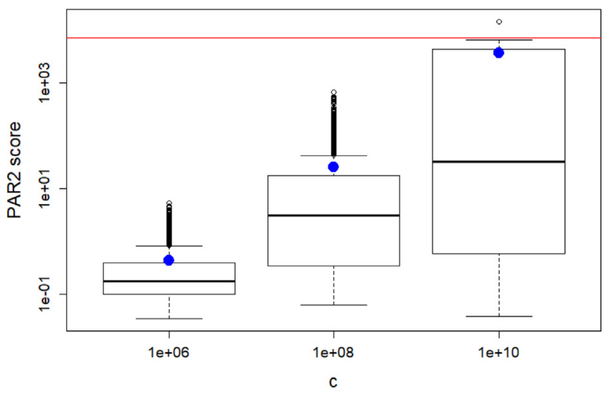


Fig. 8. PAR2 scores for different values of *c*.

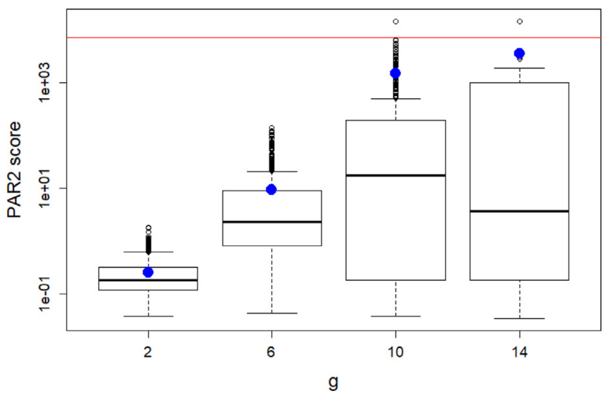


Fig. 9. PAR2 scores for different values of *g*.

6.3.2. Influence of *g*

Unlike for n and c, it is not a priori clear which influence the parameter g has on the PAR2 scores. By looking at Fig. 9, it becomes clear that the influence of g on the PAR2 scores is also more complicated than the parameters n and c. For increasing values of g, the first and the second quartile first increase and then decrease, whereas the third quartile and the average PAR2 score keep on increasing. The peaks occur at different values of g. The easiest problem instances can be found in the groups with $g = 2$ (having an average PAR2 score of 2.57×10^{-1} seconds) and $g = 6$ (having an average PAR2 score of 9.49×10^{0} seconds). All instances in these two groups could be solved within the time limit. The average PAR2 scores for $g = 10$ and $g = 14$ are much higher, namely 1.47×10^{3} seconds and 3.47×10^{3} seconds respectively.

Interestingly, we can also notice that for $g = 2$ and $g = 6$ there are respectively 520 and 198 problem instances in the dataset for

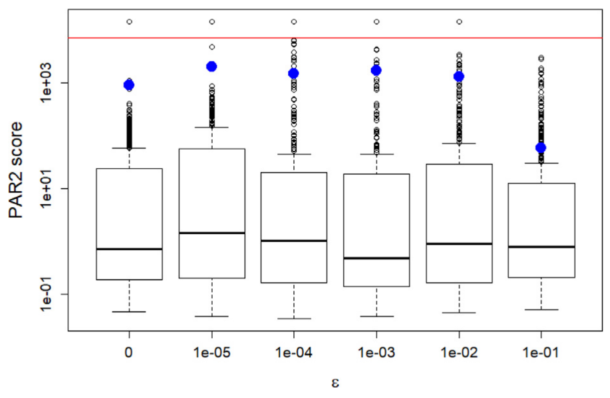


Fig. 10. PAR2 scores for different values of ϵ.

which Theorem 2 states that all items of the last group of items must belong to the optimal solution. This number is lower for the other values of g, namely 78 for $g = 10$ and 0 for $g = 14$. Hence, Theorem 2 reveals a part of the structure of the optimal solution and essentially reduces the problem instance to a smaller problem instance. The results suggest that such problem instances are also easier for ***Combo***, despite the fact that this algorithm is not based on the ideas of Theorem 2. Another potential reason why problem instances with smaller values of g are easier, is because for a fixed number of items n, the individual groups contain more items for smaller values of g. These larger groups imply that within one group, there will be more dominated items (i.e. items that have at the same time a lower profit and higher weight than another item). The existence of many dominated items can make the problem instance easier, because such items can only be part of the optimal solution if all items by which these are dominated are part of the optimal solution as well. This observation is also consistent with the conclusion that was made earlier using Fig. 6a.

6.3.3. Influence of ϵ

As can be seen in Fig. 10, the influence of ϵ on the PAR2 scores is also more complicated than the cases of n and c. The first, second and third quartile all have a different trend regarding increasing and decreasing. The average PAR2 scores for $\epsilon = 0$, $\epsilon = 10^{-5}$, $\epsilon = 10^{-4}$, $\epsilon = 10^{-3}$, $\epsilon = 10^{-2}$ and $\epsilon = 10^{-1}$ are respectively 9.07×10^{2} seconds, 1.99×10^{3} seconds, 1.49×10^{3} seconds, 1.69×10^{3} seconds, 1.28×10^{3} seconds and 5.89×10^{1} seconds. The most remarkable PAR2 score is associated with the largest value of $\epsilon = 10^{-1}$. Amongst the six different choices for ϵ, this value deviates most from the condition that ϵ is approximately 0 from Theorem 1, leading to easier problem instances. These numbers suggest that if ϵ gets too large, the problems tend to become easier, but setting $\epsilon = 0$ does not result in the hardest problem instances either. The hardest problem instances can be found in the group with $\epsilon = 10^{-5}$.

6.3.4. Influence of f and s

The PAR2 scores for different values of f and s can be found in Figs. 11 and 12. The boxplots in these figures all look very similar. The average PAR2 scores are all very close to each other: for $f = 0.1$, $f = 0.2$ and $f = 0.3$, these are respectively 1.27×10^{3} seconds, 1.18×10^{3} seconds and 1.25×10^{3} seconds, whereas for $s = 100$, $s = 200$ and $s = 300$ these are respectively 1.21×10^{3} seconds, 1.28×10^{3} seconds and 1.22×10^{3} seconds. Hence, varying these parameters does not affect the PAR2 scores a lot.

Theorems 1 and 2 are both formulated in terms of the parameters of the problem instance generator. The fact that f and s do not influence the PAR2 scores a lot, can also be understood by looking at the resulting mathematical expressions from these two

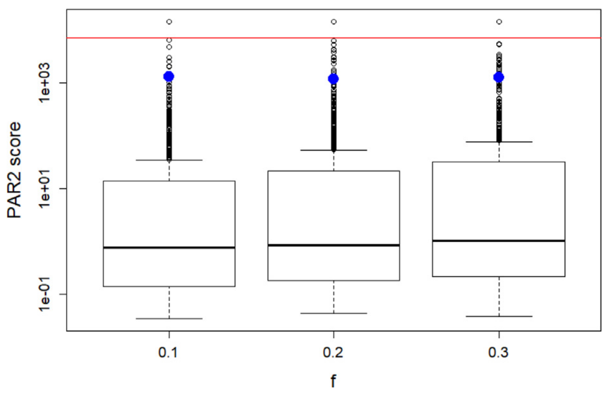


Fig. 11. PAR2 scores for different values of f.

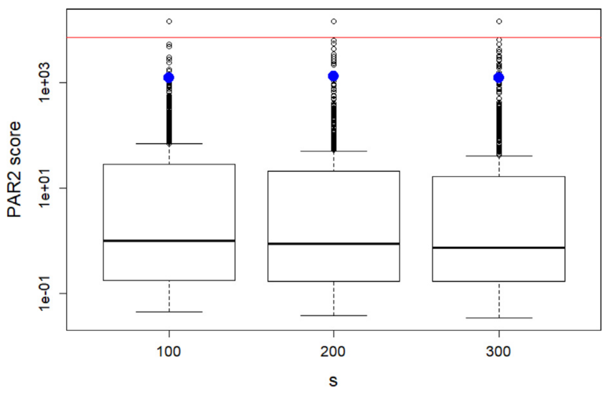


Fig. 12. PAR2 scores for different values of s.

theorems. These parameters do not occur in any dominant term (provided that the other parameters are chosen as mentioned in Section 5 and this experiment).

7. Conclusions and further work

In this paper a new class of hard problem instances for the 0–1 knapsack problem was proposed. Two theorems were proven that help explain why these problem instances are hard. These theorems characterize solutions that are hard to prune from the search space and characterize the structure of optimal solutions. A large dataset of these hard problem instances, consisting of 3240 problem instances, was generated using a full factorial design of experiments. The optimal solutions of these problem instances were computed, using a supercomputer for approximately 810 CPU-hours. A comparison with the hardest previously known instances revealed that the instances from this paper were several orders of magnitude harder, needing at least 60 times more CPU time to solve. The computational experiments also revealed that several parameters greatly influenced the hardness of the problem instances whereas the influence of other parameters was minimal.

Since the 0–1 knapsack problem is closely connected to several other problems, this work is likely to have a broader impact on other problems as well. The 0–1 knapsack problem can be seen as a special case of several packing related problems (Bonyadi et al., 2013; Kellerer et al., 2004b; Pferschy & Schauer, 2009) and it is well known that any problem which can be formulated as a binary integer linear program can be reduced to the 0–1 knapsack problem (Kellerer et al., 2004a). An important topic for future work is to investigate if the problem instances from this paper can also be adapted to create hard problem instances for these closely related problems, either as a special case or by extending them in their new context. Related to this, it could also be interesting to further investigate whether it is possible to appropriately choose the parameters of generators of existing classes for the 0–1 knapsack problem such that these generated problem instances would have similar properties as the problem instances proposed in the current paper and whether they would become harder to solve. Hard problem instances provide insights into the strengths and weaknesses of different algorithms and this knowledge can subsequently be used to create better performing algorithms. Hence, we hope that this work could lead to advancements of the state-of-the-art algorithms for multiple problems.

This work also poses a new challenge for practitioners of the 0–1 knapsack problem. It is an open question whether exact algorithms can be developed that perform better than the current state-of-the-art by exploiting the structure of the proposed problem instances. These algorithms could for example use the theorems developed in this paper. It is uncertain whether such an algorithm for the 0–1 knapsack problem would also perform better in general and thus this challenge deserves further attention.

Acknowledgments

We gratefully acknowledge the support provided by the ORDinL project (FWO-SBO S007318N, Data Driven Logistics, 1/1/2018 - 31/12/2021). This research received funding from the Flemish Government under the “Onderzoeksprogramma Artificiële Intelligentie (AI) Vlaanderen” programme. Pieter Leyman is a Postdoctoral Fellow of the Research Foundation - Flanders (FWO) with contract number 12P9419N. The computational resources and services used in this work were provided by the VSC (Flemish Supercomputer Center), funded by the Research Foundation - Flanders (FWO) and the Flemish Government - department EWI.

References

Achlioptas, D., Naor, A., & Peres, Y. (2005). Rigorous location of phase transitions in hard optimization problems. *Nature, 435*(7043), 759–764.
Applegate, D. L., Bixby, R. E., Chvátal, V., & Cook, W. (2006). *The traveling salesman problem: A computational study*. Princeton University Press.
Applegate, D. L., Bixby, R. E., Chvátal, V., Cook, W., Espinoza, D. G., Goycoolea, M., & Helsgaun, K. (2009). Certification of an optimal TSP tour through 85,900 cities. *Operations Research Letters, 37*(1), 11–15.
Balas, E., & Zemel, E. (1980). An algorithm for large zero-one knapsack problems. *Operations Research, 28*(5), 1130–1154.
Bellman, R. (1966). Dynamic programming. *Science, 153*(3731), 34–37.
Bonyadi, M. R., Michalewicz, Z., & Barone, L. (2013). The travelling thief problem: The first step in the transition from theoretical problems to realistic problems. In *2013 IEEE congress on evolutionary computation* (pp. 1037–1044). IEEE.
Caprara, A., Kellerer, H., Pferschy, U., & Pisinger, D. (2000). Approximation algorithms for knapsack problems with cardinality constraints. *European Journal of Operational Research, 123*(2), 333–345.
Cheeseman, P. C., Kanefsky, B., & Taylor, W. M. (1991). Where the really hard problems are. In *Ijcai: vol. 91* (pp. 331–337).
Chvátal, V. (1980). Hard knapsack problems. *Operations Research, 28*(6), 1402–1411.
Dantzig, G. B. (1957). Discrete-variable extremum problems. *Operations Research, 5*(2), 266–288.
Gasarch, W. I. (2019). Guest column: The third P=?NP poll. *ACM SIGACT News, 50*(1), 38–59.
Gent, I. P., & Walsh, T. (1996). The TSP phase transition. *Artificial Intelligence, 88*(1–2), 349–358.
Gu, Z., Nemhauser, G. L., & Savelsbergh, M. W. (1999). Lifted cover inequalities for 0–1 integer programs: Complexity. *INFORMS Journal on Computing, 11*(1), 117–123.
Hartmann, A. K., & Weigt, M. (2006). *Phase transitions in combinatorial optimization problems: Basics, algorithms and statistical mechanics*. John Wiley & Sons.
Hutter, F., Xu, L., Hoos, H. H., & Leyton-Brown, K. (2014). Algorithm runtime prediction: Methods & evaluation. *Artificial Intelligence, 206*, 79–111.
Jooken, J., Leyman, P., & De Causmaecker, P. (2020a). A multi-start local search algorithm for the Hamiltonian completion problem on undirected graphs. *Journal of Heuristics, 26*(5), 743–769.
Jooken, J., Leyman, P., De Causmaecker, P., & Wauters, T. (2020b). Exploring search space trees using an adapted version of Monte Carlo tree search for combinatorial optimization problems. arXiv preprint arXiv:2010.11523
Jukna, S., & Schnitger, G. (2011). Yet harder knapsack problems. *Theoretical Computer Science, 412*(45), 6351–6358.
Kellerer, H., & Pferschy, U. (1999). A new fully polynomial time approximation scheme for the knapsack problem. *Journal of Combinatorial Optimization, 3*(1), 59–71.

Kellerer, H., & Pferschy, U. (2004). Improved dynamic programming in connection with an FPTAS for the knapsack problem. *Journal of Combinatorial Optimization, 8*(1), 5–11.

Kellerer, H., Pferschy, U., & Pisinger, D. (2004a). *Knapsack problems*: vol. 57. Berlin, DE: Springer.

Kellerer, H., Pferschy, U., & Pisinger, D. (2004b). Multidimensional knapsack problems. In *Knapsack problems* (pp. 235–283). Springer.

Lawler, E. L. (1963). The quadratic assignment problem. *Management Science, 9*(4), 586–599.

Loiola, E. M., de Abreu, N. M. M., Boaventura-Netto, P. O., Hahn, P., & Querido, T. (2007). A survey for the quadratic assignment problem. *European Journal of Operational Research, 176*(2), 657–690.

Martello, S., Pisinger, D., & Toth, P. (1999). Dynamic programming and strong bounds for the 0–1 knapsack problem. *Management Science, 45*(3), 414–424.

Martello, S., & Toth, P. (1977). An upper bound for the zero-one knapsack problem and a branch and bound algorithm. *European Journal of Operational Research, 1*(3), 169–175.

Martello, S., & Toth, P. (1988). A new algorithm for the 0–1 knapsack problem. *Management Science, 34*(5), 633–644.

Mitchell, D., Selman, B., & Levesque, H. (1992). Hard and easy distributions of SAT problems. In *Aaai: vol. 92* (pp. 459–465). Citeseer.

Muñoz, M. A., Villanova, L., Baatar, D., & Smith-Miles, K. (2018). Instance spaces for machine learning classification. *Machine Learning, 107*(1), 109–147.

Pferschy, U., & Schauer, J. (2009). The knapsack problem with conflict graphs. *Journal of Graph Algorithms and Applications, 13*(2), 233–249.

Pisinger, D. (1995). An expanding-core algorithm for the exact 0–1 knapsack problem. *European Journal of Operational Research, 87*(1), 175–187.

Pisinger, D. (1997). A minimal algorithm for the 0–1 knapsack problem. *Operations Research, 45*(5), 758–767.

Pisinger, D. (2005). Where are the hard knapsack problems? *Computers and Operations Research, 32*(9), 2271–2284.

Pisinger, D., & Toth, P. (1998). Knapsack problems. In *Handbook of combinatorial optimization* (pp. 299–428). Springer.

Smith-Miles, K. (2019). MATILDA: Melbourne Algorithm Test Instance Library with Data Analytics. URL: https://matilda.unimelb.edu.au,.

Smith-Miles, K., Baatar, D., Wreford, B., & Lewis, R. (2014). Towards objective measures of algorithm performance across instance space. *Computers and Operations Research, 45*, 12–24.

Smith-Miles, K., & Bowly, S. (2015). Generating new test instances by evolving in instance space. *Computers and Operations Research, 63*, 102–113.

Smith-Miles, K., Christiansen, J., & Muñoz, M. A. (2021). "*Where are the hard knapsack problems?*" Via instance space analysis. *Computers and Operations Research, 128*, 105184.

Smith-Miles, K., van Hemert, J., & Lim, X. Y. (2010). Understanding TSP difficulty by learning from evolved instances. In *International conference on learning and intelligent optimization* (pp. 266–280). Springer.

Smith-Miles, K., & Lopes, L. (2012). Measuring instance difficulty for combinatorial optimization problems. *Computers and Operations Research, 39*(5), 875–889.