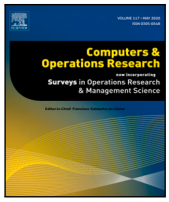
Computers & Operations Research 145 (2022) 105848



Contents lists available at ScienceDirect

Computers and Operations Research

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





Exact algorithms for the 0–1 Time-Bomb Knapsack Problem

Michele Monaci [a], Ciara Pike-Burke [b], Alberto Santini [c,*]

[a] Department of Electric, Electronic and Information Engineering, University of Bologna, Bologna, Italy
[b] Department of Mathematics, Imperial College London, London, United Kingdom
[c] Department of Economics and Business, Universitat Pompeu Fabra, Barcelona, Spain

ARTICLE INFO

Keywords:
Knapsack Problem
Stochastic optimisation
Exact algorithms
Computational experiments

ABSTRACT

We consider a stochastic version of the 0–1 Knapsack Problem in which, in addition to profit and weight, each item is associated with a probability of exploding and destroying all the contents of the knapsack. The objective is to maximise the expected profit of the selected items. The resulting problem, denoted as 0–1 Time-Bomb Knapsack Problem (01-TB-KP), has applications in logistics and cloud computing scheduling. We introduce a nonlinear mathematical formulation of the problem, study its computational complexity, and propose techniques to derive upper and lower bounds using convex optimisation and integer linear programming. We present three exact approaches based on enumeration, branch and bound, and dynamic programming, and computationally evaluate their performance on a large set of benchmark instances. The computational analysis shows that the proposed methods outperform the direct application of nonlinear solvers on the mathematical model, and provide high quality solutions in a limited amount of time.

1. Introduction

The 0–1 Knapsack Problem (01-KP) is one of the most famous problems in combinatorial optimisation. In this problem, a planner is given a set of *items*, each associated with a positive *profit* and *weight*, and a knapsack with limited *capacity*. The objective is to select a subset of items whose total weight does not exceed the knapsack capacity and whose total profit is maximal. This problem has been widely studied in the literature because of its practical and theoretical relevance, and because it arises as a subproblem in more complex problems. It is known that the 01-KP is $\mathcal{NP}$-hard, although it can be solved in pseudo-polynomial time by dynamic programming (see, e.g., Martello and Toth (1990) and Kellerer et al. (2004)).

In this paper we introduce a stochastic variant of the 0–1 Knapsack Problem, in which some items are *time-bombs*: they can explode with a given *probability* (i.e., following a Bernoulli distribution). If an item explodes, the whole content of the knapsack is lost. The objective is then to maximise the expected profit of the packed items. We call this problem the **0–1 Time-Bomb Knapsack Problem** (01-TB-KP). The interest in studying time-bomb versions of the 01-KP and its variants stems from practical applications in transporting hazardous material and in the management of data centres.

For the first application, consider a freight forwarder who has to send goods using a vehicle of fixed capacity. The forwarder can choose which deliveries to accept to maximise the total profit earned from customers without exceeding the vehicle capacity. When some of the goods to send are hazardous, this problem can be modelled as a 01-TB-KP. This may happen, for example, for lithium-ion batteries that can catch fire under unexpected mechanic stress (Farrington, 2001; Lisbona and Snee, 2011), destroying the whole content of the vehicle transporting them. In the general case, the forwarder might have an entire fleet of vehicles available; their objective is then to balance the cost of using an additional vehicle with the profits earned from additional shipments. The deterministic version of this problem, in which each vehicle corresponds to a knapsack, is known in the literature as the Fixed-Charge Multiple Knapsack Problem (FC-MKP) (Yamada and Takeoka, 2009). When considering the time-bomb version, the 01-TB-KP can be used as a subproblem to generate packings for the individual vehicles.

Similarly, a real-life application of the 01-TB-KP arises in the management of data centres (see, e.g., Srikantaiah et al. (2008), Cambazard et al. (2015) for related problems). In this case, packing items into a knapsack corresponds to allocating virtual machines to a server or applications to a container. Each customer application running on a container earns a profit, but if known vulnerabilities affect some of the applications, there is a probability that an attacker can exploit them and take over the entire container. Similar to the logistic application mentioned above, one can consider FC-MKP generalisations in which a planner must balance profits with the fixed costs incurred when spawning new containers (or buying new servers). The corresponding

* Corresponding author.
E-mail addresses: michele.monaci@unibo.it (M. Monaci), c.pikeburke@imperial.ac.uk (C. Pike-Burke), alberto.santini@upf.edu (A. Santini).

https://doi.org/10.1016/j.cor.2022.105848
Received 30 March 2021; Received in revised form 11 March 2022; Accepted 15 April 2022
Available online 11 May 2022
0305-0548/© 2022 The Author(s). Published by Elsevier Ltd. This is an open access article under the CC BY license (http://creativecommons.org/licenses/by/4.0/).

time-bomb variants can use the 01-TB-KP as a subproblem to allocate applications to the individual containers.

The paper is organised as follows. In the next section we give a formal definition of the problem and discuss similar stochastic problems related to the knapsack problem that have been considered in the literature. Section 3 introduces two relaxations of the problem that can be used to compute upper bounds on the optimal solution value, and a mathematical model that produces a heuristic solution. Section 4 presents alternative methods for computing the exact solution of the problem, including a naive enumeration approach, a branch-and-bound algorithm, and a Dynamic Programming (DP) scheme. Finally, these algorithms are computationally evaluated in Section 5 on a large benchmark of instances, comparing their performance with those of state-of-the-art solvers for Mixed Integer Non Linear Programming (MINLP).

The main contributions of the paper are the following:

- We introduce the 01-TB-KP, a stochastic variant of the classic 0–1 Knapsack Problem, that has relevant applications in logistics and data centre design. To the best of our knowledge, this is the first stochastic variant of the Knapsack Problem in which the total payoff is governed by a discrete Bernoulli distribution whose parameters depend multiplicatively on the properties of the individual items in the solution, thus filling a gap in the literature (see Section 2.1).
- We introduce a mathematical formulation for the 01-TB-KP, which uses a polynomial number of variables and constraints. We prove that the problem is weakly $\mathcal{NP}$-complete and introduce alternative exact algorithms to solve it.
- We introduce a large benchmark set of instances derived from hard 01-KP instances from the literature, and use it to compare the performance of alternative methods for solving the problem. We make both the instances and the solvers available under an open source licence (Santini, 2020).

2. Problem definition

In the 01-TB-KP we are given a knapsack with capacity $c \in \mathbb{N}$ and $n \in \mathbb{N}$ items. Each item j has a weight $w_j \in \mathbb{N}$, a profit $p_j \in \mathbb{N}$, and a probability $q_j \in [0,1)$ of exploding. As in the 01-KP, the problem requires us to determine a subset of items to pack in the knapsack whose total weight does not exceed the given capacity. However, since each item has a given probability of exploding and whenever a selected item explodes the entire content of the knapsack is lost, the objective of the problem is to maximise the total expected profit.

To simplify the notation we introduce, for each item j, the probability $\pi_j = 1 - q_j$ that item j does *not* explode, and denote by $T = \{j \in \{1,\dots,n\} \ : \ \pi_j < 1\}$ the set of time-bomb items.

The problem can be modelled using a non-linear formulation in which each binary variable x_j takes value 1 if and only if item j is selected. In addition, for notation convenience, we introduce variables $\alpha_j = 1 - q_j x_j$ (for $j \in T$) which have the following convenient property: $\alpha_j = 1 - q_j = \pi_j$ if item j is selected (i.e., $x_j = 1$) and $\alpha_j = 1$ otherwise (i.e., $x_j = 0$).

A model for the 01-TB-KP is thus:

$$\max \quad \left(\sum_{j=1}^{n} p_j x_j\right)\left(\prod_{j \in T} \alpha_j\right) \tag{1}$$

$$\text{s.t.} \quad \sum_{j=1}^{n} w_j x_j \le c \tag{2}$$

$$\alpha_j = 1 - q_j x_j \qquad j \in T \tag{3}$$

$$x_j \in \{0,1\} \qquad j \in \{1,\dots,n\} \tag{4}$$

$$\alpha_j \in \{1 - q_j, 1\} \qquad j \in T \tag{5}$$

The objective function (1), which is a polynomial of degree at most $|T| + 1$, maximises the expected profit of packed items (in the first part) taking into account that if even one of the packed items explodes, the whole profit is lost (in the second part). The knapsack capacity constraint is imposed by (2), while Eq. (3) links variables x and α, and constraints (4) and (5) define the domain of the variables.

2.1. Related problems

Stochastic Knapsack Problems (SKP) first appeared in the literature in the late 1970's (Steinberg and Parks, 1979). Typically, the term stochastic knapsack is used to define a variant of the 01-KP that incorporates some elements of stochasticity. Since there are many alternative ways of handling uncertainty, different problems have been introduced in the literature and, thus, it is difficult to give a unique definition of *the* stochastic knapsack problem.

In many problems, the set of items is known in advance, but uncertainty affects some of their characteristics, i.e., the weight (Steinberg and Parks, 1979; Morton and Wood, 1998) or the profit (Dean et al., 2008; Bhalgat et al., 2011). Most commonly, the weight is allowed to be uncertain, while the profit is either fixed or set to be a multiple of the weight. Indeed, uncertainty in the objective function can be dealt with by transforming the problem into an equivalent one in which uncertainty affects the constraints only.

In Henig (1990) the objective is to maximise the probability that the profit of the packed items exceeds a given minimum profit. In Mainville-Cohn and Barnhart (1998) a two-stage optimisation approach with recourse was used. In the first stage, items are packed without constraints, whereas in the second stage the actual weights become known, all items are collected, and a penalty is incurred for each unit of overfull capacity. The model was later extended in Merzifonluoğlu et al. (2012) to allow for earning a profit for each unit of unused capacity. In Bertsimas and Sim (2003) the robust knapsack problem is modelled using an Integer Linear Programming (ILP) formulation under the assumption that the weight of at most a given number of items can differ from the nominal value. Later, Monaci and Pferschy (2013) analyse the maximum deviation of the solution value from the optimal value of the deterministic problem in some relevant situations.

Klopfenstein and Nace (2008) consider the *chance-constrained* (CC) version of the problem, in which the capacity constraint has to be satisfied with at least a given probability. Goyal and Ravi (2010) proposed a polynomial-time approximation scheme for the CC SKP solving a linear programming reformulation of the problem, which provides tight lower bounds on the collected profit. Song et al. (2014) further expand on this, and consider both the CC knapsack and the CC set-packing problem, while Cheng et al. (2014) study the CC quadratic knapsack problem. For both the recourse and the chance-constrained SKP, Kosuch and Lisser (2010) derive upper bounds from a linear relaxation and use them in a branch and bound algorithm.

Kosuch and Lisser (2011) present a two-stage, chance-constrained SKP. In this problem, in the first stage, the planner packs the knapsack under a chance-constraint on the capacity inequality. In the second stage, once the actual weights are revealed, the planner can remove items until the capacity constraint is satisfied. Range et al. (2018) introduce a similar problem, with the difference that in the second stage the planner can accept solutions that violate the capacity constraint at the expense of a penalty that is monotonically increasing with the excess capacity.

Dean et al. (2008) consider a version of the problem in which the real weight of an item becomes known as soon as it is packed. In this problem, the planner packs items one-by-one and looks for a policy that, given the current status of the knapsack, suggests which item to attempt to pack next. If the weight of the selected item is larger than the remaining capacity, the policy terminates. Assuming access to a simulator, Pike-Burke and Grünewälder (2017) give a policy for the

SKP of (Dean et al., 2008) which is ϵ-optimal with high probability and allows for both stochastic weights and profits.

Finally, we mention that there exists a large body of literature about knapsack problems in which uncertainty is related to a temporal factor. For example, the item set can vary over time and the planner can select an item only when it materialises (see, e.g., Ross and Tsang (1989), Ross and Yao (1990), Kleywegt and Papastavrou (1998), Papastavrou et al. (1996), Kleywegt and Papastavrou (2001)), or the capacity of the knapsack can change with time (see, e.g., (He et al., 2016, 2017)). As these problems appear to be quite different from the 01-TB-KP, we do not discuss them here.

3. Upper and lower bounds for 01-TB-KP

In this section we introduce upper and lower bounds on the objective value of the 01-TB-KP. We will use these bounds when devising a branch-and-bound algorithm in Section 4.

3.1. Combinatorial upper bound

Consider a deterministic 01-KP instance in which the profit of each item j is set to $p_j\pi_j$. This problem has the same feasible set as the original problem, while its objective function overestimates the original one for each feasible solution. In particular, this objective is the expected profit when a time-bomb which explodes only causes the loss of profit for that item, while the total profit of the remaining items is unchanged. Thus, this deterministic problem is a relaxation of the 01-TB-KP, and its optimal solution value provides an upper bound $\bar{z}_1$, as formally shown in the following theorem.

Theorem 3.1. *The optimal objective value $\bar{z}_1$ of the following deterministic 01-KP*

$$\bar{z}_1 = \max \quad \sum_{j=1}^{n} p_j\pi_j x_j \tag{6}$$

$$s.t. \quad \sum_{j=1}^{n} w_j x_j \le c \tag{7}$$

$$x_j \in \{0,1\} \qquad j \in \{1,\dots,n\} \tag{8}$$

is an upper bound for the 01-TB-KP.

Proof. As already observed, any feasible solution (x,α) of model (1)–(5) defines a solution x that is feasible for (6)–(8). We have to show that, for this solution, function (6) bounds (1) from above. Indeed:

$$\left(\sum_{j=1}^{n} p_j x_j\right)\left(\prod_{j\in T}\alpha_j\right) = \sum_{j\notin T}\left(p_j x_j \prod_{j'\in T}\alpha_{j'}\right) + \sum_{j\in T}\left(p_j x_j \prod_{j'\in T}\alpha_{j'}\right) \le \tag{9}$$

$$\le \sum_{j\notin T} p_j x_j + \sum_{j\in T}\left(p_j x_j \prod_{j'\in T}\alpha_{j'}\right) \le \tag{10}$$

$$\le \sum_{j\notin T} p_j x_j + \sum_{j\in T} p_j x_j \alpha_j = \sum_{j\notin T} p_j x_j + \sum_{j\in T} p_j x_j (1-q_j x_j) = \tag{11}$$

$$= \sum_{j\notin T} p_j (1-q_j) x_j + \sum_{j\in T} p_j (1-q_j x_j) x_j = \tag{12}$$

$$= \sum_{j\notin T} p_j (1-q_j) x_j + \sum_{j\in T} p_j (1-q_j) x_j = \tag{13}$$

$$= \sum_{j=1}^{n} p_j (1-q_j) x_j, \tag{14}$$

where we pass from (10) to (11) because $\prod_{j'\in T}\alpha_{j'} \le 1$, from (11) to (12) because $\prod_{j'\in T}\alpha_{j'} \le \alpha_j$ for all $j \in T$, from (12) to (13) because $1-q_j = 1$ for $j \notin T$ and, finally, from (13) to (14) due to $x_j = x_j^2$. By the definition of $\pi_j = 1-q_j$, the last term is equal to objective function (6). □

3.2. Upper bound from the continuous relaxation

Dropping the integrality requirement (4) in model (1)–(5) yields another relaxation of the 01-TB-KP. Using the definition $\alpha_j = 1-q_j x_j$, we formulate the resulting relaxed problem in terms of the x variables only as

$$\max \ \{ f(x) \ \text{ s.t. } \ x \in P \ \},$$

where $f(x) = \left(\sum_{j=1}^{n} p_j x_j\right)\left(\prod_{j\in T}(1-q_j x_j)\right)$ is the objective function, and $P = \{x \in [0,1]^n : \sum_{j=1}^{n} w_j x_j \le c\}$ is the set of feasible solutions. In the following we will denote by $\bar{z}_2$ the optimal solution value of this relaxation.

Observe that P is a convex set and that the objective function (to be maximised) is concave, as $f(x)$ is the product of a linear function and some non-negative affine functions (Lobo et al., 1998). Thus, we can obtain an optimal solution for this relaxation using the Frank-Wolf algorithm (Frank and Wolfe, 1956). This approach is an iterative method that starts with an initial feasible solution and, at each iteration, determines an improving solution until it reaches optimality. In particular, denoting by $\bar{x}$ the current solution, one can obtain an improving direction (say $\bar{y}$) by solving the auxiliary problem

$$\bar{y} = \operatorname{argmax}\{\nabla f(\bar{x})^{\intercal} y \ \text{ s.t. } \ y \in P\}. \tag{15}$$

If $\nabla f(\bar{x})^{\intercal}(\bar{y}-\bar{x}) \le 0$ then the current solution is a local (and, hence, global) optimum. Otherwise, the next feasible solution is obtained by maximising the objective function over the segment $[\bar{x},\bar{y}]$. By definition, this ensures that the new solution improves over the current one, i.e., the method converges to an optimum. Observe that maximising f over a segment is a one-dimensional optimisation problem that can be solved using a line search approach.

In our implementation, the initial solution corresponds to the solution in which we take no items. Given $\bar{x}$, the auxiliary problem (15) corresponds to the continuous relaxation of a (deterministic) 01-KP problem in which each item j has profit $\frac{\partial f(\bar{x})}{\partial x_j}$. As such, one can use the well-known Dantzig's algorithm to find the critical item in linear time (see Balas (1980)). If the resulting solution is not optimal, our algorithm determines the next $\bar{x}$ using the line search mentioned above.

Observation. While the continuous relaxation of the deterministic 01-KP has the nice property that it includes at most one fractional item, the same does not apply for the continuous relaxation of the 01-TB-KP. Indeed, we can devise instances in which *all* items are selected at a fractional value. Consider, e.g., an instance with an even number $n > 4$ of identical items, each having profit $p_j = 1$, weight $w_j = 1$, and probability $\pi_j = \frac{1}{2}$, and let $c \ge n$ be the knapsack capacity. It is easy to show that the continuous relaxation of this instance has a unique optimal solution in which $x_j = \frac{1}{2}$ for all j.

We conclude this section with another negative property of the continuous relaxation of the 01-TB-KP. Denoting by z^* the value of an optimal 01-TB-KP integer solution, we can prove that the ratio $\bar{z}_2/z^*$ is arbitrarily large, i.e., the upper bound provided by the relaxation can be arbitrarily bad in terms of approximation.

Theorem 3.2. *The continuous relaxation of the 01-TB-KP can be arbitrarily bad.*

Proof. Consider an instance with n identical items, with $p_j = 1$, $w_j = 1$, and $q_j = q$ for all $j \in \{1,\dots,n\}$. Assume that the knapsack capacity is sufficiently large, i.e., $c \ge n$; finally, assume that $q > \frac{1}{2}$. It is easy to see that the integer optimal solution consists in taking exactly one item and that the associated optimal value is $z^* = 1-q$. Denote by $f_{n,q}(x) = \left(\sum_{j=1}^{n} x_j\right)\prod_{j=1}^{n}(1-qx_j)$ the objective function of the problem. Dropping the integrality requirement for variables x, function $f_{n,q}$ is a multi-variate continuous function, whose maximum can be derived analytically.

We first observe that, for any $i \in \{1,\dots,n\}$, the objective function can be rewritten as follows:

$$\begin{aligned} f_{n,q}(x) &= \left(\sum_{j=1}^{n} x_j\right)\prod_{j=1}^{n}(1-qx_j) \\ &= x_i\prod_{j=1}^{n}(1-qx_j) + \left(\sum_{j\neq i} x_j\right)\prod_{j=1}^{n}(1-qx_j) = \\ &= x_i(1-qx_i)\prod_{j\neq i}(1-qx_j) + \left(\sum_{j\neq i} x_j\right)(1-qx_i)\prod_{j\neq i}(1-qx_j) = \\ &= x_i\prod_{j\neq i}(1-qx_j) - qx_i^2\prod_{j\neq i}(1-qx_j) + \left(\sum_{j\neq i} x_j\right)\prod_{j\neq i}(1-qx_j) \\ &\quad - qx_i\left(\sum_{j\neq i} x_j\right)\prod_{j\neq i}(1-qx_j). \end{aligned}$$

Once $f_{n,q}$ is written in this form, we can easily compute its first partial derivatives:

$$\begin{aligned} \frac{\partial f}{\partial x_i}(x) &= \prod_{j\neq i}(1-qx_j) - 2qx_i\prod_{j\neq i}(1-qx_j) - q\left(\sum_{j\neq i} x_j\right)\prod_{j\neq i}(1-qx_j) = \\ &= \left(1 - q\sum_{j\neq i} x_j - 2qx_i\right)\prod_{j\neq i}(1-qx_j). \end{aligned}$$

For $\frac{\partial f}{\partial x_i}$ to vanish, either a term $(1-qx_j)$ or the term $\left(1 - q\sum_{j\neq i} x_j - 2qx_i\right)$ must be zero. The former case would imply that $f_{n,q}$ also vanishes and, thus, cannot correspond to a maximum. To find a candidate maximum, then, we must have

$$1 - q\sum_{j\neq i} x_j - 2qx_i = 0 \quad \forall i \in \{1,\dots,n\}. \tag{16}$$

This gives a linear system with n equations and n variables, with the unique solution $x_i = \frac{1}{q(n+1)}$ for all $i \in \{1,\dots,n\}$. This solution is feasible when $q > \frac{1}{n+1}$ and has value

$$\bar{z}_2 = \left(n\,\frac{1}{q(n+1)}\right)\left(1 - \frac{1}{n+1}\right)^n = \frac{1}{q}\left(\frac{n}{n+1}\right)^{n+1}.$$

Let us denote with $R_{n,q}$ the ratio between this value and the optimal value $z^* = 1-q$:

$$R_{n,q} = \frac{\bar{z}_2}{z^*} = \frac{1}{q(1-q)}\left(\frac{n}{n+1}\right)^{n+1},$$

and denote with R_q its value when n grows to infinity:

$$R_q = \lim_{n\to\infty}\frac{1}{q(1-q)}\left(\frac{n}{n+1}\right)^{n+1} = \frac{1}{q(1-q)}e^{-1}.$$

We obtain the desired result taking the limit of R_q when q approaches 1 from the left:

$$\lim_{q\to 1^-}\frac{1}{q(1-q)}e^{-1} = +\infty. \quad \square$$

3.3. Lower bounds

As already mentioned, the combinatorial relaxation introduced in Section 3.1 has the same feasible set as the original problem. Therefore, any feasible solution to the relaxation is also feasible for the original problem. In particular, given an optimal solution of the relaxation, we can evaluate the expected profit of the associated set of items and derive a lower bound for the 01-TB-KP. Denoting by $\hat{x}$ the optimal solution of the relaxation, this lower bound has value

$$\underline{z}_1 = \left(\sum_{j:\hat{x}_j=1} p_j\right)\left(\prod_{j:\hat{x}_j=1}\pi_j\right). \tag{17}$$

In the following, we call bound $\underline{z}_1$ the *combinatorial lower bound*.

An alternative way to get a lower bound is by solving a binary quadratic problem derived by applying Boole's inequality to the objective function. The following theorem formalises this bound, which we call the *Boole lower bound*, and proves its validity.

Theorem 3.3. *The optimal objective value of the following binary quadratic problem is a lower bound for the 01-TB-KP defined by (1)–(5).*

$$\max \quad \left(\sum_{j=1}^{n} p_j x_j\right)\left(1 - \sum_{j\in T} q_j x_j\right) \tag{18}$$

$$s.t. \quad \sum_{j=1}^{n} w_j x_j \le c \tag{19}$$

$$x_j \in \{0,1\} \qquad j \in \{1,\dots,n\} \tag{20}$$

Proof. The fact that (18) bounds (1) from below derives from the following application of Boole's inequality:

$$\begin{aligned} \prod_{j\in T}\alpha_j &= \mathbb{P}[\text{no packed TB item explodes}] \\ &= 1 - \mathbb{P}[\text{some packed TB item explode}] = \\ &= 1 - \mathbb{P}\left[\bigcup_{j\in T}(\text{packed TB item } j \text{ explodes})\right] \\ &\ge 1 - \sum_{j\in T}\mathbb{P}[\text{packed TB item } j \text{ explodes}] = \\ &= 1 - \sum_{j\in T} q_j x_j. \quad \square \end{aligned}$$

We can linearise model (18)–(20) introducing variables $z_{jk} \in \{0,1\}$ to define the product of the x variables ($z_{jk} = x_j x_k$). Thus, we obtain the following ILP formulation:

$$\max \quad \sum_{j=1}^{n} p_j x_j - \sum_{j=1}^{n}\sum_{k=1}^{n} p_j q_k z_{jk} \tag{21}$$

$$\text{s.t.} \quad \sum_{j=1}^{n} w_j x_j \le c \tag{22}$$

$$z_{jk} \ge x_j + x_k - 1 \qquad j,k \in \{1,\dots,n\} \tag{23}$$

$$x_j \in \{0,1\} \qquad j \in \{1,\dots,n\} \tag{24}$$

$$z_{jk} \in \{0,1\} \qquad j,k \in \{1,\dots,n\} \tag{25}$$

Note that constraints (23) ensure that each variable z_{jk} takes the correct values: if both x_j and x_k take value 1, then (23) forces $z_{jk} = 1$, while for the other cases the objective function ensures that z_{jk} takes value 0. While objective function (1) is a polynomial of degree $|T|+1$, the new function (21) is linear, although the model requires the definition of $\mathcal{O}(n^2)$ additional binary variables.

Theorem 3.3 states that, for any feasible solution, the value computed according to (18) underestimates the solution value. Thus, given an optimal solution to (21)–(25), we can compute a lower bound plugging the solution vector into objective function (1). The resulting solution value will be denoted as $\underline{z}_2$.

4. Exact algorithms

In this section we describe three alternative exact approaches for solving the 01-TB-KP: the first one is based on subset enumeration, the second one adopts a branch-and-bound approach, and the third one is a dynamic programming algorithm. All the three schemes assume that an oracle is available for solving the deterministic 01-KP instances.

4.1. Subset enumeration

Our first algorithm is based on the observation that the 01-TB-KP reduces to a deterministic 01-KP in case the set of time-bomb items to pack is given. Indeed, in this case, the best course of action is to maximise the profit from the non-time-bomb items (henceforth, the *deterministic items*) packed.

This suggests a solution approach that computes an optimal 01-TB-KP solution through the solution of a sequence of 01-KP instances. The scheme, reported in Algorithm 1, is as follows. For each subset of time-bomb items $S \subseteq T$, such that $\sum_{j \in S} w_j \leq c$, consider the solution obtained by (i) forcing in the solution all items in the current set S; (ii) forbidding all remaining time-bomb items (i.e., those in set $T \setminus S$); and (iii) completing the solution using some deterministic items. In particular, in the last step, we solve a deterministic 01-KP instance defined by the deterministic items and a capacity equal to $c - \sum_{j \in S} w_j$. Then, an optimal solution for the 01-TB-KP is obtained taking the best among all these solutions.

The subset enumeration approach requires the complete enumeration of all $2^{|T|}$ subsets of time-bomb items. For this reason, it can be extremely time consuming for instances in which $|T|$ is large. In any case, this approach shows that items in set T play a more prominent role than the deterministic items, a consideration that we will use in the branch-and-bound (B&B) algorithm described in the next section.

4.2. Branch-and-bound

The B&B algorithm adopts a search strategy in which branching is always associated with the inclusion or exclusion of time-bomb items. Once the subset of time-bomb items to be included in the solution has been determined, an optimal selection of the problem is obtained by solving a deterministic 01-KP instance, as in the subset enumeration scheme. According to this strategy, at each node of the branching tree, a time-bomb item can be either forced into the solution ($x_j = 1$), excluded from the solution ($x_j = 0$), or left unfixed ($x_j \in \{0, 1\}$); leaf nodes have no unfixed time-bomb items. Algorithm 2 reports pseudo-code for this algorithm.

At each node, the algorithm computes the upper and lower bounds described in Section 3. The upper bound is used to possibly prune the node, in case this value is not better than the current best solution. The lower bound value corresponds to a heuristic solution, and can be used to update the best known solution, if this gives an improvement.

If all time-bomb items are fixed (either forced or excluded), we can obtain an optimal solution value at the node by solving a 01-KP instance, in which items correspond to deterministic items only and the capacity is obtained by removing the weight of the fixed time-bomb items from the original knapsack capacity. This may lead to an update of the current best solution. We then mark the node as fully explored and backtrack to the next open node. Otherwise, if there is at least one unfixed time-bomb item, we proceed with branching. To this end, we select the unfixed time-bomb item, say j^*, with the largest p_j/w_j ratio and which fits in the residual capacity. The two children nodes correspond to either forcing or excluding j^* from the solution.

In the rest of this section, we first explain how to compute the upper and lower bounds at intermediate nodes when we must enforce branching constraints. Then, we introduce two improvements aimed at avoiding the generation of decision nodes that cannot improve the current solution and to early update the incumbent, respectively.

Algorithm 1 Subset enumeration algorithm.

function 01KP($\vec{w}, \vec{p}, c$) ▷ An oracle solving the deterministic 01-KP
  return $\max\left\{\vec{p}^\intercal \vec{x} \mid \vec{w}^\intercal \vec{x} \leq c,\ \vec{x} \in \{0,1\}^{|\vec{x}|}\right\}$
end function

function TBEnum($n, \vec{w}, \vec{p}, \vec{\pi}, c$)
  $T \leftarrow \left\{j \in \{1, \dots, n\} \mid \pi_j < 1\right\}$ ▷ Time-bomb items
  $T' \leftarrow \{1, \dots, n\} \setminus T$ ▷ Deterministic items
  $z^* \leftarrow 0$ ▷ Best solution value so far

  for $S \subseteq T$ **do** ▷ Enumerate all time-bomb item subsets
    if $\sum_{j \in S} w_j \leq c$ **then** ▷ Discard trivial cases

      $d \leftarrow$ 01KP$\left(\vec{w}|T', \vec{p}|T', c - \sum_{j \in S} w_j\right)$
      $z \leftarrow \left(d + \sum_{j \in S} p_j\right)\left(\prod_{j \in S} \pi_j\right)$
      if $z > z^*$ **then**
        $z^* \leftarrow z$ ▷ Update the best solution value
      end if
    end if
  end for

  return z^*
end function

Algorithm 2 Branch & Bound algorithm for the 01-TB-KP.

function TBBranchBound($n, \vec{w}, \vec{p}, \vec{\pi}, c$)
  $T \leftarrow \left\{j \in \{1, \dots, n\} \mid \pi_j < 1\right\}$ ▷ Time-bomb items
  $T' \leftarrow \{1, \dots, n\} \setminus T$ ▷ deterministic items
  $S \leftarrow \emptyset$ ▷ Time-bomb items forced in the solution
  $\bar{S} \leftarrow \emptyset$ ▷ Time-bomb items excluded from the solution
  $z^* \leftarrow 0$ ▷ Value of the current best feasible solution

  ExploreNode($T, S, \bar{S}, z^*$)
  return z^*
end function

procedure ExploreNode($T, S, \bar{S}, z^*$)
  Compute lower $\underline{z}$ and upper $\bar{z}$ bounds for the residual instance with:
    item set $\{1, \dots, n\} \setminus (S \cup \bar{S})$ ▷ Unfixed items
    knapsack capacity $c - \sum_{j \in S} w_j$ ▷ Residual capacity

  if $\underline{z} > z^*$ **then**
    $z^* \leftarrow \underline{z}$ ▷ Update the best solution value
  end if
  if $\bar{z} \leq z^*$ **then**
    Prune the node
  end if

  if $T \setminus (S \cup \bar{S}) = \emptyset$ **then** ▷ All time-bomb items fixed
    $d \leftarrow$ 01KP$\left(\vec{w}|T', \vec{p}|T', c - \sum_{j \in S} w_j\right)$
    $z \leftarrow \left(d + \sum_{j \in S} p_j\right)\left(\prod_{j \in S} \pi_j\right)$
    if $z > z^*$ **then**
      $z^* \leftarrow z$ ▷ Update the best solution value
    end if
  else
    Choose a time-bomb item $j^* \in T \setminus (S \cup \bar{S})$ fitting in the residual capacity ▷ Branching
    ExploreNode($S \cup \{j^*\}, \bar{S}, z^*$) ▷ Force j^*
    ExploreNode($S, \bar{S} \cup \{j^*\}, z^*$) ▷ Exclude j^*
  end if
end procedure

4.2.1. Bounding

We now discuss how to compute local lower and upper bounds at each node of the branch-and-bound tree, i.e., when branching conditions force or forbid some time-bomb items in the solution. In the following, we denote as $S \subseteq T$ and $\bar{S} \subseteq T$, respectively, the set of time-bomb items forced ($x_j = 1$) or forbidden ($x_j = 0$) in the solution. In addition, we denote with $F = T \setminus \left(S \cup \bar{S}\right)$ the set of time-bomb items that are free. i.e., not already fixed by branching conditions.

We start the analysis from the combinatorial upper bound, $\bar{z}_1$ described in Section 3.1. Imposing branching conditions requires to solve formulation (6)–(8) with additional constraints

$$x_j = 1 \ (j \in S) \quad \text{and} \quad x_j = 0 \ (j \in \bar{S}) \tag{26}$$

This can be done considering a deterministic knapsack instance defined by time-bomb items in set F and by deterministic items, and a reduced capacity $c_S = c - \sum_{j \in S} w_j$. Let us denote by $\bar{z}_1^F$ the value of the resulting problem. In addition, let $p_S = \sum_{j \in S} p_j$ and $q_S = \prod_{j \in S}(1-q_j)$ be the sum of the profits and the product of the probabilities of not exploding of the items forced in the knapsack, respectively. We can obtain an upper bound for our problem as $\bar{z}_1 = \left(p_S + \bar{z}_1^F\right) q_S$, as shown by the following theorem.

Theorem 4.1. *A valid upper bound on the optimal solution value of the 01-TB-KP in which items $S \subseteq T$ are packed and items $\bar{S} \subseteq T$ are not packed is given by*

$$\bar{z}_1 = \left(p_S + \bar{z}_1^F\right) q_S. \tag{27}$$

Proof. Let R be the set of items that do not belong to S and are selected in an optimal solution of formulation (6)–(8) with additional constraints (26). Let us denote by p_R and q_R the corresponding sum of profits and product of probabilities, respectively, and denote by $\bar{z}^R = p_R q_R$. The objective value of a 01-TB-KP solution composed by items $S \cup R$ is

$$z^{S \cup R} = \left(p_S + p_R\right) q_S q_R.$$

We have

$$\begin{aligned}
\bar{z}_1 &= \left(p_S + \bar{z}_1^F\right) q_S \geq \\
&\geq \left(p_S + \bar{z}^R\right) q_S = \\
&= \left(p_S + p_R q_R\right) q_S = \\
&= p_S q_S + p_R q_R q_S \geq \\
&\geq p_S q_S q_R + p_R q_R q_S = \\
&= \left(p_S + p_R\right) q_S q_R = z^{S \cup R},
\end{aligned}$$

where the first inequality is due to the definition of $\bar{z}_1^F$, and the second one follows from Theorem 3.1 applied to the instance of the 01-TB-KP with time-bomb item set F and capacity c_S. This result shows that $\bar{z}_1$ is a valid upper bound on the value of any feasible solution for the current subproblem. □

We observe that the solution of formulation (6)–(8) with additional constraints (26) produces a set of items whose total weight is not larger than the knapsack capacity, i.e., it allows the computation of a valid lower bound at the node.

The same consideration applies to lower bound $\underline{z}_2$, which can be computed adding branching conditions (26) to formulation (21)–(25), and plugging the resulting solution into the objective function (1).

Finally, for the continuous relaxation upper bound $\bar{z}_2$, we observe that branching conditions can be easily handled by simply adding these constraints to the definition of the feasible set P. As a consequence, the initial solution consists of the items in S. In addition, at each iteration, the improving direction $\bar{y}$ to be determined must satisfy branching conditions (26) as well, i.e., time-bomb items in S and $\bar{S}$ are fixed a-priori to 1 and to 0, respectively, when solving the continuous relaxation of the deterministic knapsack problem required by (15).

4.2.2. Early pruning

In contrast to the deterministic 01-KP, in the 01-TB-KP it may happen that adding an item to a solution reduces the value of the objective function. Indeed, the total profit of the selected items is multiplied by the probabilities of each selected item not to explode; hence, adding an item with high probability of exploding and low profit, may produce a decrease of the objective function value. More formally, we prove the following result.

Theorem 4.2. *Consider a feasible solution containing item set S and let $j \notin S$ be an item such that $\sum_{i \in S} w_i + w_j \leq c$. Then adding item j increases the solution value only if and only if*

$$\frac{\pi_j}{1-\pi_j} p_j > \sum_{i \in S} p_i. \tag{28}$$

Proof. Adding item j does not increase the solution value if

$$\begin{aligned}
& \sum_{i \in S} p_i \prod_{i \in S} \pi_i \geq \left(\sum_{i \in S} p_i + p_j\right) \left(\prod_{i \in S} \pi_i\right) \pi_j \\
\Longrightarrow \quad & \sum_{i \in S} p_i \geq \left(\sum_{i \in S} p_i + p_j\right) \pi_j \\
\Longrightarrow \quad & (1-\pi_j) \sum_{i \in S} p_i \geq \pi_j p_j \\
\Longrightarrow \quad & \sum_{i \in S} p_i \geq \frac{\pi_j}{1-\pi_j} p_j
\end{aligned}$$

where the last condition derives from the non-negativity of probabilities. □

The above result provides a condition under which adding an item to a feasible solution does *not* lead to an improved solution. Let S be the set of time-bomb items fixed in the solution at some node of the branch-and-bound tree. According to Theorem 4.2, we can prune all descendant nodes associated with an item j that do not satisfy condition (28), if any. Thus, given such an item j, we branch generating a unique descendant node corresponding to excluding the item. It is worth noting that, if item j does not satisfy condition (28) associated with item set S, then the same will happen also for the that item j and an enlarged item set $S \cup \{k\}$ (for each item k). In other words, as all the profits are positive, there exists no set of items $S' \supset S$ such that adding j to S' improves the solution value.

We can thus use this result to prune a large amount of nodes associated with the inclusion of items with high probability of exploding. Indeed, as one may expect, the closer π_j to 1, the more likely that condition (28) holds; conversely, an item with π_j close to 0, i.e., with high probability of exploding, has a low probability of being profitable and improving a given solution. In the special case $\pi_j = 1/2$, the condition reduces to $p_j > \sum_{i \in T} p_i$, i.e., the profit of item j must be larger than the sum of the profits of all items included in the current solution.

4.2.3. Early bounding

The B&B algorithm described in Section 4.2 solves a 01-KP induced by deterministic items at each leaf node, i.e., whenever all time-bomb items have been fixed by branching (line 21 to line 25). The scheme can be improved by computing feasible solutions at intermediate nodes of the branching tree as well, to increase the probability of finding high-quality solutions early in the search. Noting that the solution computed in this way does not change when excluding a time-bomb item, in our implementation we solve a 01-KP associated with the deterministic items only at nodes generated by fixing a time-bomb item in the solution (line 28 in Algorithm 2). Finally, observe that this policy corresponds to anticipating the computation of the 01-KP solutions, but does not increase the total number of such computations. However, the knowledge of a near-optimal solution from the beginning of the search may considerably reduce the computational effort for solving an instance to optimality, as shown by the computational experiments reported in Section 5.

4.3. Dynamic programming

We present here an algorithm to solve the 01-TB-KP based on Dynamic Programming (DP) recursion. In the following, we assume for convenience that the time-bomb items are indexed by $T = \{1, \dots, t\}$ and that an upper bound $U \in \mathbb{N}$ on the total profit of time-bomb items

that fit in the knapsack (disregarding any probability to explode) is available. Solving a deterministic 01-KP with items in set T, capacity c and omitting probabilities π_j can provide such a bound.

Let us denote with $d \in \{0, \dots, c\}$ the capacity used by time-bomb items in an optimal solution; accordingly, the total weight of the deterministic items is at most $c - d$. One can find an optimal solution to the 01-TB-KP by guessing the value of d, finding the best subset of time-bomb items in the "sub-knapsack" of capacity d, and the best subset of deterministic items in the "sub-knapsack" of capacity $c - d$. The full algorithm, thus, uses three ingredients: (i) a DP algorithm to pack time-bomb items; (ii) another DP algorithm to pack deterministic items; and (iii) a linear scan to combine the two partial solutions for all possible values of d.

For the first algorithm (packing the time-bomb items), define the quantity $\bar{\pi}(d, v, j)$ as the highest probability of not exploding for a solution using only time-bomb items of total weight at most d, total profit (disregarding the probabilities) at least v, and using the first j items only (i.e., those indexed by $\{1, \dots, j\}$). We use the convention that $\bar{\pi}(d, v, j) = 0$ if no such solution exists. The DP recursion is the following:

$$\bar{\pi}(d, v, j) = \max\left\{\bar{\pi}(d, v, j-1), \bar{\pi}(d - w_j, v - p_j, j-1)\,\pi_j\right\}, \tag{29}$$

where we assume that entries with negative $d - w_j$ or $v - p_j$ take value zero. The first term corresponds to *not* packing item j, while the second term corresponds to packing it. Initial values are $\bar{\pi}(d, 0, 0) = 1$ for all $d \in \{0, \dots, c\}$, and $\bar{\pi}(d, v, 0) = 0$ for all $d \in \{0, \dots, c\}$ and $v \geq 1$.

For the second algorithm (packing the deterministic items), we use the classical recursion of Bellman (1954) and Dantzig (1957). We denote with $z(c-d)$ the profit of the optimal solution of the 01-KP over the deterministic items, with a knapsack of capacity $c - d$.

Then, solving the 01-TB-KP amounts to finding

$$z^* = \max_{d=0,\dots,c} \; \max_{v=0,\dots,U} \left\{ \left[v + z(c-d)\right] \bar{\pi}(d, v, t) \right\}. \tag{30}$$

where values $z(c - d)$ are in the DP table of the second problem, and values $\bar{\pi}(d, v, t)$ are in the DP table of the first problem.

This algorithm also answers an important question about the complexity of 01-TB-KP, which we formalise in the following theorem.

Theorem 4.3. *The 01-TB-KP is weakly $\mathcal{NP}$-hard.*

Proof. It is clear that the 01-TB-KP is $\mathcal{NP}$. In addition, it cannot be solved in polynomial time because it generalises the 01-KP, arising as a special case when $T = \emptyset$. To prove that the problem can be solved in pseudo-polynomial time, we analyse the complexity of the Dynamic Programming algorithm proposed above. One can compute all the entries of matrix $\bar{\pi}$ in $O(c\,U\,t)$ time. Let $p_{\max} = \max_{j \in T} p_j$ and note that $U \leq n p_{\max}$. Because $t \leq n$, then the complexity of computing all entries of matrix $\bar{\pi}$ is bounded by $O(n^2\, c\, p_{\max})$. All the entries $z(c-d)$ can be computed in time $O(n\,c)$, while the final scan (30) takes $(c+1)(U+1) = O(c\,n\,p_{\max})$ evaluations. Thus, the complexity for calculating the entries of matrix $\bar{\pi}$ dominates that of the other steps of the algorithm, and the total complexity of the approach is $O(n^2\, c\, p_{\max})$, i.e., the algorithm runs in pseudo-polynomial time. □

5. Computational experiments

We implemented all proposed algorithms in C and we evaluated them through a computational analysis on a large set of benchmark of instances. Unless explicitly specified, all the experiments were executed on an Intel Xeon processor running at 1.7 GHz.

Our branch-and-bound algorithm uses Combo Martello et al. (1999) to solve the deterministic knapsack subproblems. For computing lower bound z_2 described in Section 3.3, we solved the quadratic model (18)–(20) and the linear model (21)–(25) using the commercial solver Gurobi 9.0. All codes are publicly available under an open source licence, on GitHub (Santini, 2020).

5.1. Instance generation

To assess the performances of the algorithms in different situations, we generated a large set of instances with different characteristics. Our benchmark is composed of 5 classes of problems: the first four classes are based on the *hard instances* introduced by Pisinger (2005) for the 01-KP, whereas the last one is designed to challenge some of the bounds introduced in Section 3. Each class includes instances of different sizes, in terms of number of items; in particular, we generated problems with $n \in \{100, 500, 1000, 5000\}$.

Class 1 The instances of this class are defined using weight, profit and capacity values of hard 01-KP instances, without modifying them. Given a 01-KP instance and an input parameter $B \in [0, 1]$, we first determine the $\lceil nB \rceil$ items with the largest profit, and define these items as time-bombs (set T). Then, we define the following parameters

$$\overline{P} = \max_{j \in T} \{p_j\},$$
$$\underline{P} = \min_{j \in T} \{p_j\},$$
$$p = \max_{j \notin T} \left\{p_j \;:\; p_j < \underline{P}\right\}.$$

$\overline{P}$ represents the largest profit value over all time-bomb items, whereas $\underline{P}$ is the smallest profit of a time-bomb item, and p is the largest profit value of a deterministic item that is strictly smaller than $\underline{P}$. Finally, we generate the probability of exploding of each time-bomb item using the following formula

$$q_j = 0.1\,\frac{p_j - p}{\overline{P} - p} \quad (j \in T). \tag{31}$$

In this way, each time-bomb item has a low probability of exploding ($q_j \leq 0.1$), modelling the realistic scenarios in logistic and data centre management mentioned in Section 1. In addition, probabilities are proportional to profits, preventing situations in which higher-profit items have a lower probability of exploding.

We generated instances with $B \in \{0.1, 0.2, 0.5\}$. For each pair (n, B) we defined 10 problems, that were obtained randomly selecting different 01-KP base instances.

Class 2 These instances are similar to those of Class 1, but we determine the time-bomb items selecting the $\lceil nB \rceil$ ones with the largest profit-to-weight ratio p_j/w_j. Accordingly, we define $\overline{P}$, $\underline{P}$, and p in terms of profit-to-weight ratios and replace p_j by p_j/w_j for computing probabilities in (31).

Class 3 These instances are identical to those of Class 1 but for probabilities, which are not correlated with the profits nor the weights of the items. Rather, for each time-bomb item j, we set

$$q_j \sim \mathcal{B}(1, 10), \tag{32}$$

where $\mathcal{B}(b_1, b_2)$ denotes a beta distribution of parameters b_1 and b_2. The use of a beta distribution with such parameters produces time-bomb items with a small probability of exploding.

Class 4 These instances are identical to those of Class 2 but for probabilities, which are generated according to (32).

Class 5 In these instances, all items share the same weight $w_j = w$, while profits and probabilities are defined in such a way that the product $p_j \pi_j$ is a constant. These instances are intended to be challenging, as all items appear to be identical when solving the combinatorial upper bound $\bar{z}_1$ described in Section 3.1. To define hard knapsack problems, for a given capacity value c, the weights are defined as

$$w = \left\lfloor k\,\frac{c}{n} \right\rfloor, \tag{33}$$

where k is a parameter. Profits are strongly correlated to weights as follows

$$p_j = \max\{1, \lceil w + \varepsilon \rceil\}, \quad \varepsilon \sim \mathcal{N}\left(0, \frac{w}{8}\right), \tag{34}$$

where $\mathcal{N}(\mu, \sigma)$ denotes a normal distribution of mean μ and variance σ^2. We use the max to ensure that each profit take a positive (integer) value, since the normal distribution has unbounded support and the noise term ε could, in principle, make p_j negative. Finally, to achieve constant $p_j \pi_j$ product, we define $\pi_j = p/p_j$ (where $p = \min_{j \in \{1,\dots,n\}} p_j$), i.e., almost all items in these instances are time-bomb. We generated instances with $k \in \{1.6, 2.0, 2.4\}$, and defined 10 problems for each pair (n, k), using a different random seed for each instance.

Each of the five classes includes 120 instances. Our complete benchmark set, composed of 600 problems, is publicly available on GitHub (Santini, 2020).

5.2. Upper and lower bound comparison

Our first set of experiments is aimed at evaluating the computational performance of the lower and upper bounds described in Section 3. Denoting by $\bar{z}$ the value of an upper bound on a given instance of the problem, the associated percentage gap is defined as

$$\%\text{gap} = 100\,\frac{\bar{z} - z^*}{z^*},$$

where z^* denotes the optimal solution value for the instance (whenever unknown, this value is replaced by the best known solution value). Similarly, the percentage gap associated with a lower bound $\underline{z}$ is

$$\%\text{gap} = 100\,\frac{z^* - \underline{z}}{z^*}.$$

Table 1 reports, for each bound, the average percentage gap (column **%gap**), the average computing time in seconds (column **time**), and the percentage of instances in which the bound matches with the optimal or best known solution (column **%opt**). We omit the times for the combinatorial lower bound $\underline{z}_1$ because they are always negligible, given an optimal solution of relaxation (6)–(8).

Regarding bound $\underline{z}_2$, we first ran preliminary experiments to compare the performances of the quadratic model (18)–(20) with those of its linearised version (21)–(25). The results showed that solver Gurobi 9.0 was always more efficient on the quadratic model than on the linearised one; for this reason, all results refer to the model (18)–(20). However, in many cases, the computing time needed for computing a provably optimal solution of the model was too large to embed this lower bound within a B&B algorithm. Thus, we analysed the performance of the solver with different time limits, namely 1 s, 10 s, and 1 h.

The table shows that lower bound $\underline{z}_2$ usually outperforms $\underline{z}_1$ in terms of both average gap and number of optimal solutions found. Bound $\underline{z}_1$, however, is much faster to be computed as it only requires to evaluate objective function (1) for the solution returned by the combinatorial relaxation described in Section 3.1. Quite interestingly, even with a time limit of 1 s, $\underline{z}_2$ gives a smaller %gap and a larger %opt than $\underline{z}_1$. For larger time limits, the quality of $\underline{z}_2$ improves even further; with a time limit of 1 h, an optimal solution is found in almost 70% of the instances.

With respect to upper bounds, the upper bound from the continuous relaxation, $\bar{z}_2$, consistently outperforms the combinatorial upper bound, $\bar{z}_1$, in terms of average gaps, often by two orders of magnitude; in addition, the associated upper bound coincides with the optimal solution value in a larger number of instances. Though larger than those needed for computing $\bar{z}_1$, computing times for $\bar{z}_2$ are usually under 1 s, i.e., they are still acceptable for usage within a B&B algorithm.

Finally, note that, as expected, instances of Class 5 are by far the most challenging in our benchmark, all bounds having their worst performances on the instances in this class.

5.3. Tuning of the branch-and-bound algorithm

We now compare the performances of the B&B algorithm in alternative configurations, obtained activating/deactivating some relevant features of the algorithm. We run all the experiments using a time limit equal to 3600 s per instance and compared the following variants:

- **SE:** The Subset Enumeration scheme described in Section 4.1.
- **V1:** The B&B algorithm introduced in Section 4.2, embedding bounds $\bar{z}_1$ and $\underline{z}_1$ and implementing the early pruning and early bounding acceleration techniques described in Sections 4.2.2 and 4.2.3, respectively.
- **V2:** The algorithm obtained adding lower bound $\underline{z}_2$ to **V1**; in our implementation, we solve the quadratic model with a time limit of 10 s at the root node. In addition, we solve the model with time limit equal to 1 s every 1000 nodes of the enumeration tree.
- **V3:** The algorithm obtained adding upper bound $\bar{z}_2$ to **V1**, i.e., also computing at each node the continuous relaxation of the mathematical formulation of the problem by means of the procedure described in Section 3.2.
- **V4:** The algorithm obtained adding both $\underline{z}_2$ and $\bar{z}_2$ to the **V1**, and executing the lower bound procedure according to the same setting described above.
- **V5:** this algorithm is obtained from **V4** by disabling the early pruning described in Section 4.2.2.
- **V6:** this algorithm is obtained from **V4** by disabling the early bounding described in Section 4.2.3.

For each algorithm, we report the following information:

- **%gap**: average percentage gap. Denoting by $\bar{z}$ and $\underline{z}$ the best upper and lower bound found by the algorithm on a given instance, the percentage gap is computed as

  $$\%\text{gap} = 100\,\frac{\bar{z} - \underline{z}}{\bar{z}};$$

- **time**: average computing time (in seconds);
- **nds**: average number of B&B nodes explored;
- **%opt**: number of instances solved to proven optimality.

For variant **SE**, we do not report the average number of nodes, as this algorithm does not compute dual bounds allowing for pruning, and hence it always explores $2^{|T|}$ nodes. Similarly, we do not report the percentage gap, which cannot be computed for lack of dual bounds.

The results in Table 2 show that, as may be expected, our B&B algorithm outperforms the basic scheme **SE**. Version **V1**, that includes the computation of bounds $\bar{z}_1$ and $\underline{z}_1$, solves most of the instances with $n \leq 1000$, except those in Class 5. Adding lower bound $\underline{z}_2$ produces some worsening of the results: version **V2** has, on average, larger gaps and smaller percentage of instances solved to optimality than **V1**. On the contrary, adding upper bound $\bar{z}_2$ gives considerable improvements: indeed, version **V3** is able to solve more than 98% of the instances within an average computing time under 3 min. The situation is similar when adding to **V1** both $\underline{z}_2$ and $\bar{z}_2$: though the results of version **V4** are comparable to those of **V3** in terms of number of optimal solutions, the former is better in terms of average percentage gap, computing time, and number of nodes.

Finally, results for versions **V5** and **V6** show that deactivating the corresponding features produces a considerable worsening of the approach. Indeed, the average percentage gap grows from 0.18% to 0.90% when deactivating early pruning, and the number of instances solved to proven optimality drops from 98.67% to 88.17% when removing early bounding.

Table 1
Comparison of lower and upper bounds.

		$\underline{z}_1$		$\underline{z}_2$ (1 s)			$\underline{z}_2$ (10 s)			$\underline{z}_2$ (1 h)			$\bar{z}_1$			$\bar{z}_2$		
Class	Size	%gap	%opt	%gap	time	%opt	%gap	time	%opt	%gap	time	%opt	%gap	time	%opt	%gap	time	%opt
Class 1	100	9.78	50.00	0.21	0.63	76.67	0.10	1.22	83.33	0.10	0.81	83.33	10.59	0.00	46.67	0.23	0.01	26.67
	500	48.78	23.33	66.67	0.89	30.00	7.53	5.19	80.00	0.00	732.76	93.33	19.48	0.00	23.33	0.03	0.01	43.33
	1000	41.52	40.00	66.67	0.89	26.67	53.33	7.60	40.00	0.00	971.28	90.00	19.30	0.00	40.00	0.03	0.01	43.33
	5000	50.58	43.33	100.00	1.00	0.00	100.00	10.00	0.00	0.00	1105.04	80.00	18.68	0.05	43.33	0.01	6.61	40.00
	All sizes	37.66	39.17	58.39	0.85	33.33	40.24	6.00	50.83	0.03	702.47	86.67	17.01	0.01	38.33	0.07	1.66	38.33
Class 2	100	31.05	13.33	0.70	0.76	73.33	0.00	2.61	90.00	0.00	2.78	90.00	10.73	0.00	13.33	0.20	0.00	16.67
	500	52.91	16.67	46.67	0.72	36.67	0.33	4.12	70.00	0.01	351.50	73.33	15.22	0.00	16.67	0.08	0.00	16.67
	1000	63.02	13.33	66.70	0.90	20.00	36.68	7.29	50.00	0.01	507.43	86.67	16.34	0.00	13.33	0.04	0.12	20.00
	5000	79.31	10.00	100.00	1.00	0.00	100.00	10.00	0.00	2.76	1338.94	70.00	11.88	0.07	10.00	0.00	0.10	16.67
	All sizes	56.57	13.33	53.52	0.85	32.50	34.25	6.01	52.50	0.69	550.17	80.00	13.54	0.02	13.33	0.08	0.05	17.50
Class 3	100	22.71	43.33	0.17	0.42	83.33	0.07	1.10	83.33	0.07	0.77	83.33	11.81	0.00	43.33	0.27	0.00	36.67
	500	43.62	36.67	40.00	0.80	50.00	10.00	5.23	66.67	0.06	509.85	73.33	17.68	0.00	36.67	0.06	0.03	43.33
	1000	49.59	36.67	70.00	0.95	16.67	43.33	6.50	46.67	0.00	814.32	70.00	17.59	0.00	36.67	0.03	0.12	30.00
	5000	50.05	40.00	100.00	1.00	0.00	100.00	10.00	0.00	6.48	1171.85	80.00	21.38	0.01	40.00	0.13	0.10	33.33
	All sizes	41.49	39.17	52.54	0.79	37.50	38.35	5.71	49.17	1.65	624.20	76.67	17.12	0.00	39.17	0.12	0.06	35.83
Class 4	100	44.11	16.67	3.75	0.90	70.00	0.02	2.33	83.33	0.01	1.51	90.00	11.16	0.00	16.67	0.18	0.00	3.33
	500	82.37	3.33	60.00	0.93	23.33	7.26	5.68	50.00	0.29	961.37	66.67	20.65	0.00	3.33	0.04	0.07	10.00
	1000	74.42	10.00	66.67	0.91	26.67	60.00	7.91	30.00	0.03	630.07	86.67	11.80	0.01	10.00	0.02	0.01	20.00
	5000	93.26	3.33	100.00	1.00	0.00	100.00	10.00	0.00	0.02	1590.59	66.67	13.02	0.14	3.33	0.00	0.12	16.67
	All sizes	73.54	8.33	57.60	0.94	30.00	41.82	6.48	40.83	0.09	795.89	77.50	14.16	0.04	8.33	0.06	0.05	12.50
Class 5	100	100.00	0.00	4.84	0.91	3.33	4.84	1.65	3.33	4.84	0.75	3.33	91.75	0.00	0.00	0.55	0.02	0.00
	500	100.00	0.00	84.08	1.00	3.33	5.09	9.80	6.67	5.09	617.01	6.67	98.04	0.00	0.00	0.73	0.08	23.33
	1000	100.00	0.00	93.85	1.00	0.00	49.92	10.00	0.00	4.87	863.26	3.33	99.19	0.00	0.00	1.45	0.13	16.67
	5000	100.00	0.00	100.00	1.00	0.00	100.00	10.00	0.00	0.42	3600.00	93.33	97.01	0.00	0.00	0.92	0.17	93.33
	All sizes	100.00	0.00	70.69	0.98	1.67	39.96	7.86	2.50	3.81	1270.25	26.67	96.50	0.00	0.00	0.91	0.10	33.33
Overall		61.85	20.00	58.55	0.88	27.00	38.93	6.41	39.17	1.25	788.59	69.50	31.66	0.01	19.83	0.25	0.39	27.50

Table 2
Comparison of Subset Enumeration and different B&B versions to solve the 01-TB-KP.

		SE		V1				V2				V3				V4				V5				V6			
Type	Size	time	%opt	%gap	time	nds	%opt	%gap	time	nds	%opt	%gap	time	nds	%opt	%gap	time	nds	%opt	%gap	time	nds	%opt	%gap	time	nds	%opt
Class 1	100	1095.28	70.00	2.66	299.04	50023085.9	93.33	4.72	366.29	4046200.5	90.00	0.00	0.85	44.2	100.00	0.00	1.96	38.3	100.0	0.00	1.80	38.3	100.00	0.00	2.28	38.3	100.00
	500	3600.00	0.00	15.82	1081.89	47853858.3	70.00	16.41	1375.69	594013.5	66.67	0.00	7.55	166.0	100.00	0.00	11.73	166.0	100.0	0.00	7.86	166.0	100.00	0.00	96.75	24803.7	100.00
	1000	3567.88	3.33	17.37	1082.21	20247003.8	70.00	17.37	1122.78	97693.7	70.00	0.00	2.61	73.6	100.00	0.00	7.91	73.6	100.0	0.00	8.32	73.6	100.00	0.00	36.71	5158.1	100.00
	5000	3600.00	0.00	17.62	1584.43	3602219.0	56.67	18.16	1731.35	282141.1	56.67	0.01	198.64	297.3	96.67	0.01	197.58	297.6	96.7	0.01	218.13	297.5	96.67	2.57	374.05	2082.1	93.33
	All sizes	2965.79	18.33	13.37	1011.89	30431541.8	72.50	14.17	1149.03	1255012.2	70.83	0.00	52.41	145.3	99.17	0.00	54.79	143.9	99.2	0.00	59.03	143.8	99.17	0.64	127.45	8020.5	98.33
Class 2	100	1119.57	70.00	1.00	271.20	929217.1	93.33	1.59	487.92	145394.9	86.67	0.00	0.78	2539.5	100.00	0.00	5.15	2528.6	100.0	0.00	14.86	2528.6	100.00	0.00	4.71	2528.6	100.00
	500	3480.03	3.33	12.33	1051.56	18587685.4	73.33	12.64	1111.69	284814.2	70.00	0.00	104.75	90647.7	100.00	0.00	130.29	36552.2	96.7	0.00	132.21	18820.5	96.67	0.00	143.66	54053.7	96.67
	1000	3600.00	0.00	12.94	1216.45	1073564.8	66.67	12.95	1276.65	301615.4	66.67	0.00	31.38	2152.5	100.00	0.00	66.77	2152.5	100.0	0.00	41.35	1968.5	100.00	0.00	312.51	47122.8	100.00
	5000	3600.00	0.00	10.71	2007.36	130794.4	46.67	11.01	2085.79	22257.5	46.67	0.00	172.90	1273.9	100.00	0.00	224.53	1273.9	100.0	0.00	304.03	1273.9	100.00	25.32	2315.41	23890.4	43.33
	All sizes	2949.90	18.33	9.25	1136.64	5180315.4	70.00	9.55	1240.51	188520.5	67.50	0.00	77.45	24153.4	100.00	0.00	106.68	10626.8	99.2	0.00	123.11	6147.9	99.17	6.33	694.07	31898.9	85.00
Class 3	100	1086.34	70.00	0.00	0.26	82562.0	100.00	0.00	13.04	81939.9	100.00	0.00	0.03	34.9	100.00	0.00	0.80	21.7	100.0	0.00	0.86	21.7	100.00	0.00	0.80	21.7	100.00
	500	3600.00	0.00	13.31	840.66	80857403.4	76.67	13.68	900.42	220884.7	76.67	0.00	21.83	121.8	100.00	0.00	10.84	121.1	100.0	0.00	15.92	120.9	100.00	0.00	55.46	687.3	100.00
	1000	3600.00	0.00	14.55	1088.12	26692000.7	70.00	15.95	1358.49	376034.9	63.33	0.85	173.43	239.3	96.67	0.71	143.09	405.9	96.7	0.80	156.37	326.1	96.67	2.40	347.41	13132.2	93.33
	5000	3600.00	0.00	20.10	1571.21	8907304.3	60.00	20.88	1707.71	51679.2	56.67	0.22	260.25	760.6	96.67	0.15	236.96	1580.7	96.7	0.13	230.55	1787.6	96.67	19.00	772.20	6051.6	80.00
	All sizes	2971.59	17.50	11.99	875.06	29134817.6	76.67	12.63	994.92	182634.7	74.17	0.27	113.88	289.1	98.33	0.21	97.92	532.4	98.3	0.23	100.92	564.1	98.33	5.35	293.97	4973.2	93.33
Class 4	100	1226.04	66.67	0.00	0.55	185086.7	100.00	0.00	42.66	182652.7	100.00	0.00	0.10	169.6	100.00	0.00	3.48	130.4	100.0	0.00	3.20	130.4	100.00	0.00	3.50	131.0	100.00
	500	3600.00	0.00	16.00	981.80	8005582.9	73.33	17.22	1201.00	288627.2	70.00	0.88	264.62	3039.2	96.67	0.29	256.66	3268.6	96.7	1.84	293.81	7551.8	93.33	5.06	536.86	63737.1	93.33
	1000	3600.00	0.00	8.08	635.79	17406260.2	83.33	9.09	909.43	154356.0	80.00	0.00	5.64	490.5	100.00	0.00	13.22	490.5	100.0	0.00	17.08	490.5	100.00	13.62	1281.77	198982.5	83.33
	5000	3600.00	0.00	9.82	1712.55	1592982.4	56.67	9.92	1996.53	59451.2	53.33	0.00	495.52	1926.0	100.00	0.00	406.57	1926.0	100.0	0.00	565.17	1916.7	96.67	79.22	2895.74	22565.9	20.00
	All sizes	3006.51	16.67	8.47	832.68	6797478.1	78.33	9.06	1037.41	171271.8	75.83	0.22	191.47	1406.3	99.17	0.07	169.98	1453.9	99.2	0.46	219.82	2522.3	97.50	24.48	1179.47	71354.1	74.17
Class 5	100	3600.00	0.00	70.15	3081.88	482647463.6	23.33	91.86	3600.00	62357584.0	0.00	0.00	17.38	1108.5	100.00	0.00	16.05	514.0	100.0	0.00	16.06	562.3	100.00	0.00	17.38	562.3	100.00
	500	3600.00	0.00	99.26	3600.00	143076350.2	0.00	98.85	3600.00	8821396.3	0.00	0.00	180.68	4064.3	100.00	0.00	138.23	2916.1	100.0	0.00	126.42	8270.9	100.00	0.00	206.58	10268.5	100.00
	1000	3600.00	0.00	99.63	3600.00	111730081.6	0.00	99.63	3600.00	8031406.3	0.00	1.50	657.32	15366.5	93.33	0.77	584.82	15323.4	96.7	13.02	1571.06	64801.8	70.00	16.71	1526.87	98533.9	66.67
	5000	3600.00	0.00	97.01	3600.00	66253571.1	0.00	97.01	3600.00	1558698.5	0.00	2.05	269.64	1271.7	93.33	1.66	268.75	1308.4	93.3	2.20	267.94	1447.8	93.33	2.20	268.24	1436.9	93.33
	All sizes	3600.00	0.00	91.51	3470.47	200926866.6	5.83	96.84	3600.00	20192271.3	0.00	0.89	281.26	5452.7	96.67	0.61	251.96	5015.5	97.5	3.81	495.37	18770.7	90.83	4.73	504.77	27700.4	90.00
Overall		3098.76	14.17	26.92	1465.35	54494203.9	60.67	28.45	1604.37	4397942.1	57.67	0.27	143.29	6289.4	98.67	0.18	136.27	3554.5	98.7	0.90	199.65	5629.8	97.00	8.31	559.94	28789.4	88.17

5.4. Comparison of alternative exact methods

Our last set of experiments compares our B&B algorithm in its best tuning (i.e., activating all bounding procedures, version **V4**) with alternative exact solution methods. The first such method is the dynamic programming algorithm described in Section 4.3. In addition, we evaluate the direct application of general-purpose solvers for nonlinear programming on the mathematical formulation (1)–(5). The model has some nice properties, e.g., its continuous relaxation asks to maximise a concave function over a convex set. Thus, modern commercial solvers, that nowadays include sophisticated tools (preprocessing, heuristics, domain reduction techniques, ...) could perform competitively, at least on small instances. We use solvers Couenne and Baron, which are state-of-the-art for the solution of MINLPs.

The results are shown in Table 3, whose columns have the same meaning as in Table 2. The Dynamic Programming algorithm runs with a memory limit equal to 8 GB; when this limit is reached, the algorithm halts. Since this limit allowed to run Dynamic Programming only on small instances, we grouped the instances according to their size. In addition, we do not report column **%gap** for this algorithm, as it either computes an optimal solution or terminates because it runs out of memory. Because of licencing restrictions, experiments with Baron and Couenne were performed on a different machine equipped with a (slightly) faster processor, namely an Intel Xeon running at 2.53 GHz. Because Couenne often reported incorrect solutions due to numerical instabilities, for this solver we report an additional column (**%valid**) that gives the percentage of instances for which the run was normally completed. Average values for percentage gap, computing time and percentage of optimal solutions are computed considering the valid instances only.

The results show that Couenne is able to solve only a small number of instances, with a percentage gap more than 30% on average. Though

Table 3
Comparison of exact methods to solve the 01-TB-KP: solvers Couenne and Baron, Dynamic Programming and B&B.

	B&B			DP		Baron			Couenne			
Size	%gap	time	%opt	time	%opt	%gap	time	%opt	%gap	time	%opt	%valid
100	0.00	5.49	100.00	2642.58	45.33	15.50	946.78	74.67	7.84	1684.64	54.36	99.33
500	0.06	109.55	98.67	–	–	19.70	1372.78	64.00	19.33	2565.82	32.89	99.33
1000	0.29	163.16	98.67	–	–	21.22	1607.87	59.33	20.25	2636.10	30.60	89.33
5000	0.36	266.88	97.33	–	–	66.45	3461.52	10.00	78.67	2976.03	17.33	100.00
Overall	0.18	136.27	98.67	2642.58	45.33	30.72	1847.24	52.00	31.89	2462.03	33.85	97.00

having a similar percentage gap on average, Baron is able to solve more than half of the instances to optimality, with an average computing time of 30 min. Indeed, both solvers have satisfactory performances for small and medium instances, while they are quite inefficient when facing instances with $n = 5000$. Our B&B algorithm clearly outperforms the solvers: the number of instances solved to proven optimality is considerably larger than that of the solvers, and the average computing time and percentage gap are reduced by more than an order of magnitude.

6. Conclusions

In this paper we studied the 0–1 Time-bomb Knapsack problem, a stochastic variant of the well-known 0–1 Knapsack Problem, in which items have an associated probability of exploding. We presented a natural mathematical model for this problem and introduced procedures, based on combinatorial arguments and on convex optimisation, for computing lower and upper bounds on the optimal solution value. Finally, we presented alternative schemes for the exact solution of the problem, and tested them computationally on a large benchmark of instances. The comparison, which also involves general purpose solvers that are state-of-the-art for nonlinear programming, show that our branch-and-bound algorithm is the most efficient way to attack the problem.

Future work shall analyse time-bomb versions of other problems belonging to the class of knapsack problems. For example, the real-life scenarios mentioned in Section 1 suggest that the time-bomb versions of the Fixed-Charge Knapsack Problem and of the Multiple Knapsack Problem are worth studying due to their practical relevance.

CRediT authorship contribution statement

Michele Monaci: Conceptualization, Formal analysis, Investigation, Methodology, Software, Writing – original draft, Writing – review & editing. **Ciara Pike-Burke:** Formal analysis, Investigation, Methodology, Writing – review & editing. **Alberto Santini:** Conceptualization, Formal analysis, Investigation, Methodology, Software, Writing – original draft, Writing – review & editing.

Acknowledgement

The work of Michele Monaci was supported by the Air Force Office of Scientific Research, United States under award number FA8655-20-1-7012.

References

Balas, E., 1980. An algorithm for large zero-one knapsack problems. Oper. Res. 28 (5), 1130–1154.
Bellman, R., 1954. Some applications of the theory of dynamic programming – a review. Oper. Res. 2 (3), 275–288.
Bertsimas, D., Sim, M., 2003. The price of robustness. Oper. Res. 52, 35–53.
Bhalgat, A., Goel, A., Khanna, S., 2011. Improved approximation results for stochastic knapsack problems. In: Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms, SODA '11. SIAM, pp. 1647–1665.
Cambazard, H., Mehta, D., O'Sullivan, B., Simonis, H., 2015. Bin packing with linear usage costs. ArXiv arXiv:1509.06712.
Cheng, J., Delage, E., Lisser, A., 2014. Distributionally robust stochastic knapsack problem. SIAM J. Optim. 24 (3), 1485–1506.
Dantzig, G., 1957. Discrete variable extremum problems. Oper. Res. 5 (2), 266–277.
Dean, B., Goemans, M., Vondrák, J., 2008. Approximating the stochastic knapsack problem: the benefit of adaptivity. Math. Oper. Res. 33 (4), 945–964.
Farrington, M., 2001. Safety of lithium batteries in transportation. J. Power Sources 96 (1), 260–265.
Frank, M., Wolfe, P., 1956. An algorithm for quadratic programming. Nav. Res. Logist. 3, 95–110.
Goyal, V., Ravi, R., 2010. A PTAS for the chance-constrained knapsack problem with random item sizes. Oper. Res. Lett. 38 (3), 161–164.
He, Y., Wang, X., Li, W., Zhao, S., 2017. Exact algorithms and evolutionary algorithms for randomized time-varying knapsack problem. J. Softw. 28, 185–202.
He, Y., Zhang, X., Li, W., Li, X., Wu, W., Gao, S., 2016. Algorithms for randomized time-varying knapsack problems. J. Comb. Optim. 31, 95–117.
Henig, M., 1990. Risk criteria in a stochastic knapsack problem. Oper. Res. 38 (5), 820–825.
Kellerer, H., Pferschy, U., Pisinger, D., 2004. Knapsack Problems. Springer, Berlin, Germany.
Kleywegt, A., Papastavrou, J., 1998. The dynamic and stochastic knapsack problem. Oper. Res. 46 (1), 17–35.
Kleywegt, A.J., Papastavrou, J.D., 2001. The dynamic and stochastic knapsack problem with random sized items. Oper. Res. 49 (1), 26–41.
Klopfenstein, O., Nace, D., 2008. A robust approach to the chance-constrained knapsack problem. Oper. Res. Lett. 36 (5), 628–632.
Kosuch, S., Lisser, A., 2010. Upper bounds for the 0-1 stochastic knapsack problem and a B&B algorithm. Ann. Oper. Res. 176 (1), 77–93.
Kosuch, S., Lisser, A., 2011. On two-stage stochastic knapsack problems. Discrete Appl. Math. 159 (16), 1827–1841.
Lisbona, D., Snee, T., 2011. A review of hazards associated with primary lithium and lithium-ion batteries. Process Saf. Environ. Prot. 89 (6), 434–442.
Lobo, M.S., Vandenberghe, L., Boyd, S., Lebret, H., 1998. Applications of second-order cone programming. Linear Algebra Appl. 284, 193–228.
Mainville-Cohn, A., Barnhart, C., 1998. The stochastic knapsack problem with random weights: A heuristic approach to robust transportation planning. In: Proceedings of the Triennial Symposium on Transportation Analysis. pp. 1–13.
Martello, S., Pisinger, D., Toth, P., 1999. Dynamic programming and strong bounds for the 0-1 knapsack problem. Manage. Sci. 45 (3), 414–424.
Martello, S., Toth, P., 1990. Knapsack Problems: Algorithms and Computer Implementations. Wiley.
Merzifonluoğlu, Y., Geunes, J., Romeijn, E., 2012. The static stochastic knapsack problem with normally distributed item sizes. Math. Program. 134 (2), 459–489.
Monaci, M., Pferschy, U., 2013. On the robust knapsack problem. SIAM J. Optim. 23 (4), 1956–1982.
Morton, D., Wood, K., 1998. On a stochastic knapsack problem and generalizations. In: Woodroof, D. (Ed.), Advances in Computational and Stochastic Optimization, Logic Programming, and Heuristic Search. Springer, pp. 149–168.
Papastavrou, J.D., Rajagopalan, S., Kleywegt, A.J., 1996. The dynamic and stochastic knapsack problem with deadlines. Manage. Sci. 42 (12), 1706–1718.
Pike-Burke, C., Grünewälder, S., 2017. Optimistic planning for the stochastic knapsack problem. In: Singh, A., Zhu, J. (Eds.), International Conference on Artificial Intelligence and Statistics. pp. 1114–1122.
Pisinger, D., 2005. Where are the hard knapsack problems? Comput. Oper. Res. 32 (9), 2271–2284.
Range, T.M., Kozlowski, D., Petersen, N.C., 2018. A shortest-path-based approach for the stochastic knapsack problem with non-decreasing expected overfilling costs. Comput. Oper. Res. 97, 111–124.
Ross, K., Tsang, D., 1989. The stochastic knapsack problem. IEEE Trans. Commun. 37 (7), 740–747.
Ross, K., Yao, D., 1990. Monotonicity properties for the stochastic knapsack. IEEE Trans. Inform. Theory 36 (5), 1173–1179.
Santini, A., 2020. Code and instances for the 01-TB-KP. Github repository. URL https://github.com/alberto-santini/tbkp/.
Song, Y., Luedtke, J., Küçükyavuz, S., 2014. Chance-constrained binary packing problems. INFORMS J. Comput. 26 (4), 735–747.
Srikantaiah, S., Kansal, A., Zhao, F., 2008. Energy-aware consolidation for cloud computing. In: Isaacs, R., Zhou, Y. (Eds.), Proceedings of the 2008 Usenix Annual Technical Conference.
Steinberg, E., Parks, M., 1979. A preference order dynamic program for a knapsack problem with stochastic rewards. J. Oper. Res. Soc. 30 (2), 141–147.
Yamada, T., Takeoka, T., 2009. An exact algorithm for the fixed-charge multiple knapsack problem. European J. Oper. Res. 192 (2), 700–705.