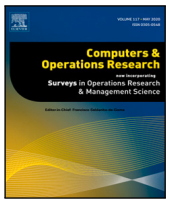
Computers & Operations Research 178 (2025) 107010

Contents lists available at ScienceDirect

Computers and Operations Research

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



Model-based algorithms for the 0-1 Time-Bomb Knapsack Problem

Roberto Montemanni [a,*], Derek H. Smith [b]

[a] Department of Sciences and Methods for Engineering, University of Modena and Reggio Emilia, Via Amendola, 2, 42122 Reggio Emilia, Italy
[b] Computing and Mathematics, University of South Wales, Pontypridd CF37 1DL, Wales, UK

ARTICLE INFO

Keywords:
0-1 Knapsack
Time-bomb
Model-based algorithms
Mixed integer linear programming

ABSTRACT

A stochastic version of the 0–1 Knapsack Problem recently introduced in the literature and named the 0–1 Time-Bomb Knapsack Problem is the topic of the present work. In this problem, in addition to profit and weight, each item is characterized by a probability of exploding, and therefore destroying all the contents of the knapsack, in case it is loaded. The optimization aims at maximizing the expected profit of the selected items, which takes into account also the probabilities of explosion, while fulfilling the capacity constraint. The problem has real-world applications in logistics and cloud computing.

In this work, two model-based algorithms are introduced. They are based on partial linearizations of a non-linear model describing the problem. Extensive computational results on the instances available in the literature are presented to position the new methods as the best-performing ones, while comparing against those previously proposed.

1. Introduction

The 0–1 Knapsack Problem (01-KP) is a classic problem in combinatorial optimization, which has been extensively studied in the past decades due to its importance in many practical applications. In this problem, a set of items is given, each characterized by a profit and a weight, together with a capacity. The problem aims at selecting a subset of items to be loaded into the knapsack in such a way that the sum of the weights of the selected items does not exceed the knapsack capacity, while the sum of the profits is maximized. The problem is $\mathbb{NP}$-hard, although dynamic programming can solve it in pseudo-polynomial time (see Martello and Toth, 1990). An extended overview of deterministic knapsack problems and their variations can be found in the recent two-parts survey (Cacchiani et al., 2022a,b).

The subject of the present study is a variant of the 01-KP recently introduced by Monaci et al. in Monaci et al. (2022), called the 0–1 Time-Bomb Knapsack Problem (01-TB-KP). In this variant, some of the items are time-bombs: they explode with a given probability, and in case of explosion of an item inserted in the knapsack, the whole content of the knapsack gets destroyed and the total profit becomes 0. The objective is the maximization of the *expected* profit, therefore taking into account the risk of explosion of the items. The problem has practical relevance and applications in hazardous material transportation (e.g. lithium–ion batteries, see Farrington, 2001; Lisbona and Snee, 2011). Another real-life application can be identified in the management of data centers (see Srikantaiah et al., 2008), where the knapsack model represents the allocation of virtual machines to a server or applications to a container. Each application running on a container earns a profit, but if it accidentally creates some vulnerability that can be exploited by hackers, there is a probability that an intruder can take over the entire container, voiding the entire profit. Various methods are proposed in Monaci et al. (2022) for the 01-TB-KP. Namely, relaxations of the problem to provide upper bounds; a mathematical model operating on a simplified version of the problem and able to provide effective heuristic solutions; exact solving methods ranging from a naive enumeration approach, a dynamic programming scheme and a tailored branch and bound framework with several alternative branching strategies, a non-linear model handled by ah-hoc solvers.

Optimization involving uncertain information is gaining popularity in the community due to its capability of taking into account unpredictable factors that inherently characterize the real world (Donati et al., 2003). The 01-TB-KP is part of this trend. It is a stochastic optimization problem, although it differs from the several stochastic/probabilistic versions of the knapsack problem modeled before in the literature. Stochastic knapsack problems track back to the late 1970's (Steinberg and Parks, 1979), and uncertainty can affect either the weight (Morton and Wood, 1998) or the profit (Dean et al., 2008). A two stage stochastic optimization was adopted as a paradigm in Mainville-Cohn and Barnhart (1998) and Merzifonluoglu et al. (2012) to attack the problem. To remain in the field of uncertain optimization, Bertsimas and Sim (2003) introduced a robust knapsack

* Corresponding author.
E-mail addresses: roberto.montemanni@unimore.it (R. Montemanni), derek.smith@southwales.ac.uk (D.H. Smith).

https://doi.org/10.1016/j.cor.2025.107010
Received 27 September 2024; Received in revised form 4 February 2025; Accepted 5 February 2025
Available online 13 February 2025

problem under the assumption that the weight of some items can differ from the standard value. In the same context, Monaci and Pferschy (2003) analyzed the maximum deviation of the solution value from the optimal profit of the deterministic problem under certain hypothesis. A version of the knapsack problem where the weights of the items are revealed once they are packed is presented in Dean et al. (2008). Another enriched version of the knapsack problem relevant for the present study is the quadratic knapsack. A review on this problem can be found in Galli et al. (2025b), while information on a new extension taking into account setup costs for items of different classes, can be found in Galli et al. (2025a).

In this paper we exploit a property of logarithms to devise two efficient model-based algorithms for the 01-TB-KP. Computational results confirm the superiority of the new approaches when compared to the other methods available in the literature.

The paper is organized as follows. In Section 2 the 01-TB-KP is formally introduced, while a non-linear model is described in Section 3. The two model-based algorithms, that represent the main contribution of the paper, are presented and analyzed in Section 4. Computational experiments, covering all the instances available from the previous literature, are presented and commented in Section 5. Conclusions and a proposal for future work are presented in Section 6.

2. Problem definition

The 01-TB-KP can be formalized as follows. A knapsack with capacity $c \in \mathbb{N}^+$ is given, together with a set N composed of n items. Each item $k \in N$ is characterized by a weight $w_k \in \mathbb{N}^+$, a profit $p_k \in \mathbb{N}^+$, and a probability $q_k \in (0, 1]$ of *not* exploding. The objective is to select a subset of items to be packed in the knapsack in such a way that the total weight does not exceed the given capacity c, while the total *expected* profit is maximized. The expected profit $\mathbb{E}(K)$ of a set of items $K \subseteq N$ can be expressed in a closed form as follows (Monaci et al., 2022):

$$\mathbb{E}(K) = \left(\sum_{k \in K} p_k\right)\left(\prod_{k \in K} q_k\right) \tag{1}$$

where the sum accounts for the profits of the objects of K and the product calculates the *probability of survival* (not exploding) for the objects of K.

3. A non-linear model

The following non-linear model, first presented in Monaci et al. (2022), fully defines the 01-TB-KP. It will be at the basis of the algorithms that we will propose in Section 4. There is a set of binary variables x such that $x_k = 1$ if the object $k \in N$ is selected, 0 otherwise.

$$(NL)\ \max \left(\sum_{k \in N} p_k x_k\right)\left(\prod_{k \in K} q_k x_k\right) \tag{2}$$

$$s.t. \sum_{k \in N} w_k x_k \leq c \tag{3}$$

$$x_i \in \{0, 1\} \qquad i \in N \tag{4}$$

The objective function (2) implements the calculation defined by (1). The constraint (3) expresses the capacity constraint, while in (4) the domain of the variables is defined.

From a computational viewpoint, the bottleneck of this model (NL) is represented by the non-linearity of the objective function (2), which prevents the use of the very efficient off-the-shelf linear programming solvers available. In Section 4 we will propose a workaround technique to mitigate the effects of non-linearity.

4. Algorithms

The methods we propose are iterative and are based on the following well-known property of logarithms, already used in the past to linearize concepts within linear programming optimization (Montemanni et al., 2008):

$$\prod_{k \in K} \alpha_k = \beta \iff \sum_{k \in K} \log(\alpha_k) = \log(\beta) \tag{5}$$

This property allows the solution of the 01-TB-KP as a sequence of integer linear programming models, leading to efficient and effective algorithmic solving strategies.

4.1. An algorithm based on the optimization of profits

A first algorithm can be devised by optimizing the total profit collected, while moving the probability of survival to a constraint, and looping until the optimality of an incumbent solution is proven, in an iterative fashion. The best 01-TB-KP solution encountered during the computation (calculated according to (1)) is stored in the variable $BestHeuValue$, while the variable $LastExactProfit$ contains the objective function value of the last problem $P^{(i)}$, $i < j$, of the sequence solved to optimality.

$$\left(P^{(j)}\right)\ \max \sum_{k \in N} p_k x_k \tag{6}$$

$$s.t. \sum_{k \in N} w_k x_k \leq c \tag{7}$$

$$\sum_{k \in N} \log(q_k) x_k \geq \log\left(\frac{BestHeuValue}{LastExactProfit}\right) \tag{8}$$

$$\sum_{k \in N : \hat{x}_k^{(i)} = 1} x_k + \sum_{k \in N : \hat{x}_k^{(i)} = 0} (1 - x_k) \leq n - 1 \qquad i = 1, 2, \ldots, j-1 \tag{9}$$

$$x_i \in \{0, 1\} \qquad i \in N \tag{10}$$

The objective function (6) aims at maximizing the profit collected, here without taking into account the probability of survival. Constraint (7) models capacity (it is the same as (3) but we repeat it, together with (4), for completeness in these models). The next two sets of constraints are those implementing the iterative mechanism. Constraint (8) forces the solution to have a probability of survival greater than (or equal to) that of the logarithm of the ratio between $BestHeuValue$ — the real value of the best solution retrieved so far — and $LastExactProfit$ — the objective function value of the last problem $P^{(i)}$ solved to optimality. The logic behind this constraint is as follows. Considering that the value of (6) will be lower than or equal to that found in the previous iteration solved to optimality, in order to get solution values not lower than $BestHeuValue$, we need a probability greater than or equal to the ratio. An arbitrarily small constant can be added to the r.h.s. of the constraint to have solutions with an objective (6) strictly higher than $LastExactProfit$. This would theoretically speed up the convergence of the method, however this might generate issues related to numerical precision, so we avoid this solution in our implementation. Constraints (9) finally force the solution of the model to be different from the solutions retrieved in the previous iterations, by imposing that at least one variable flips its value. The domain of the variables is formally described by constraints (10).

The model is solved iteratively within an algorithm that can be summarized in the pseudocode presented in Algorithm 1.

Algorithm 1 takes as input a 01-TB-KP instance and returns a feasible solution together with an upper bound for the optimal solution value. In case the value of the feasible solution matches the bound, the solution is proven optimal. At line 1–4 variables are initialized. BestHeuSol contains the best 01-TB-KP solution retrieved so far, and BestHeuValue is its cost; LastExactProbability contains the value of the last model $P^{(i)}$ solved to optimality in the previous iterations; j is a counter. The model $P^{(0)}$ is built at line 5. The main loop is then entered,

Algorithm 1: IterativeAlgorithmP

```
Data: A 01-TB-KP instance
Result: A solution and an upper bound for the optimal solution value
1  BestHeuSol=∅;
2  BestHeuValue=0;
3  LastExactProfit=+∞;
4  j = 0;
5  Build model P^(0);
6  while BestHeuValue<LastExactProfit and computational time < MT do
7      Solve model P^(j) with a maximum computation time of ML seconds;
8      if P^(j) has been solved to optimality then
9          if P^(j) is feasible then
10             LastExactProfit=Value of the optimal solution of P^(j);
11         else
12             LastExactProfit=BestHeuValue;
13         end
14     end
15     if BestHeuValue<LastExactProfit then
16         x̂^(j)= Retrieved solution of P^(j);
17         if Eq. (2) evaluated on x̂^(j) >BestHeuValue then
18             BestHeuSol=x̂^(j);
19             BestHeuValue=Value(x̂^(j));
20         end
21         j = j + 1;
22         Build model P^(j);
23     end
24 end
25 return BestHeuSol, LastExactProfit;
```

and the computation will continue until either the optimality of a 01-TB-KP solution is proven or the maximum allowed computation time of MT seconds has elapsed. At each iteration, the current model $P^{(j)}$ is solved, allowing a maximum computation time of ML seconds (line 7). In case the model is solved to optimality (line 8), if the solution is feasible, then at line 10 the variable LastExactProfit is updated to the value of the optimal solution of $P^{(j)}$. If the model is instead proven infeasible (line 11) the search is completed and the optimal solution for the 01-TB-KP has been retrieved, since a solution potentially improving the current best one does not exist. Notice that LastExactProfit is set to BestHeuValue in order to force the end of the computation (line 12). In case the computation continues (line 15), we save the current solution of $P^{(j)}$ into $\hat{x}^{(j)}$ at line 16, and in case the real profit of such a solution improves the best known 01-TB-KP solution retrieved so far, we update the values of variables BestHeuSol and BestHeuValue (lines 18–19). At lines 21–22 the counter j is finally incremented by one unit and the new problem $P^{(j)}$ is created (notice that it is enough to modify $P^{(j-1)}$ by updating the right-hand side of constraint (8) and adding a new constraint (9)). The best solution retrieved together with an upper bound for the optimal solution value are returned at line 25, to terminate the computation.

In order to speed up the computation, it is possible to calculate an initial lower bound prior to running Algorithm 1 by solving a 01-KP on non-bomb items. Variables $BestHeuSol$ and $BestHeuValue$ can therefore be initialized consequently. We add this step for the computational experiments presented in Section 5, where the extra time required by the calculation of the lower bound will be accounted for.

4.2. An algorithm based on the optimization of probabilities

With respect to the method discussed in Section 4.1, in this approach we maximize the survival probability while we impose a constraint on the total profit collected, in an iterative context as before. The model on which the method is based in the following one, where the meaning of $BestHeuValue$ remains the same, while the variable $LastExactProfit$ is substituted by $LastExactProbability$, which will contain the objective function value of the last problem $S^{(i)}$, $i < j$, of the sequence solved to optimality.

$$\left(S^{(j)}\right) \quad \max \sum_{k \in N} \log(q_k) x_k \tag{11}$$

$$s.t. \sum_{k \in N} w_k x_k \leq c \tag{12}$$

$$\sum_{k \in N} p_k x_k \geq \frac{BestHeuValue}{LastExactProbability} \tag{13}$$

$$\sum_{k \in N : \hat{x}_k^{(i)}=1} x_k + \sum_{k \in N : \hat{x}_k^{(i)}=0} (1 - x_k) \leq n - 1 \qquad i = 1, 2, \ldots, j-1 \tag{14}$$

$$x_i \in \{0; 1\} \qquad i \in N \tag{15}$$

The objective function (11) aims at maximizing the probability of survival of the solution, without taking into account the profits of the items. Constraint (12) models capacity. Analogously to model $P^{(j)}$, the next two sets of constraints are those implementing the iterative mechanism. Constraint (13) forces the solution to have a total profit higher than that of the ratio between $BestHeuValue$ — the real value of the best solution retrieved so far — and $LastExactProbability$ — the objective function value of the last problem $P^{(i)}$ solved to optimality. The logic behind this constraint is as follows. In order to improve a heuristic solution with cost $BestHeuValue$, and considering that the value of (11) will be lower than or equal to that found in the previous iteration solved to optimality, we need a total profit greater than the ratio. Also in this case an arbitrarily small constant can be added to the r.h.s. of constraint (13) to produce solutions with a value of (11) strictly higher than $LastExactProbability$, but in our implementation we prefer not to insert it, in order to eliminate any possible issue related to numerical precision. Constraints (14) finally force the solution of the model to be different from the solutions retrieved in the previous iterations, as seen in Section 4.1. The domain of the variables is formally described by constraints (15).

The overall method can be summarized by the pseudocode presented in Algorithm 2.

Algorithm 2 takes as input a 01-TB-KP instance and returns a feasible solution together with a variable indicating whether the solution has been proven to be exact or if it is simply heuristic. No upper bound is returned by this algorithm, due to its characteristics. At lines 1–4 variables are initialized, then the model $S^{(0)}$ is built at line 5. The main loop is then entered, and the computation continues until the maximum allowed computation time MT has elapsed (line 6) or when a solution is proven to be optimal (line 12). At each loop the current model $S^{(j)}$ is solved, allowing a maximum computation time of ML seconds (line 7). In case the model has been solved to optimality (line 8), if the solution is feasible, then at line 10 the variable LastExactProbability is updated to the value of the optimal solution of $S^{(j)}$. If the model is instead proven infeasible (line 11) the search is completed and the optimal solution has been retrieved (line 12). If the computation continues, we save the current solution of $S^{(j)}$ into $\hat{x}^{(j)}$ at line 15, and we try to increment the set of selected items by solving a deterministic 01 knapsack on the residual capacity and considering only non-bomb items. More objects can potentially be inserted since problem $S^{(j)}$ only considers probabilities, and adding non-bomb items does not change the overall probability. The eventual new items selected are saved into variables $\hat{y}^{(j)}$, and $\hat{z}^{(j)}$ is defined as the elementwise *or* operator between the previous two sets of variables

Algorithm 2: IterativeAlgorithmS

Data: A 01-TB-KP instance
Result: A solution and a value indicating whether the solution is optimal or heuristic

1 BestHeuSol=∅;
2 BestHeuValue=0;
3 LastExactProbability=1;
4 $j = 0$;
5 Build model $S^{(0)}$;
6 **while** *Computational time* $< MT <$ *MaxTime* **do**
7 Solve model $S^{(j)}$ with a maximum computation time of ML seconds;
8 **if** $S^{(j)}$ *has been solved to optimality* **then**
9 **if** $P^{(j)}$ *is feasible* **then**
10 LastExactProbability=Value of the optimal solution of $S^{(j)}$;
11 **else**
12 **return** BestHeuSol,"exact";
13 **end**
14 **end**
15 $\hat{x}^{(j)}$=Optimal solution of $S^{(j)}$;
16 $\hat{y}^{(j)}$=Optimal solution of the deterministic Knapsack problem with non-bomb unselected items and residual capacity;
17 $\hat{z}^{(j)} = \hat{x}^{(j)} \vee \hat{y}^{(j)}$;
18 **if** *Eq.* (2) *evaluated on* $\hat{z}^{(j)} >$*BestHeuValue* **then**
19 BestHeuSol=$\hat{z}^{(j)}$;
20 BestHeuValue=Value($\hat{z}^{(j)}$);
21 **end**
22 $j = j + 1$;
23 Build model $S^{(j)}$;
24 **end**
25 **return** BestHeuSol,"heuristic";

(lines 16–17). In case the real profit of solution $\hat{z}^{(j)}$ improves the best known 01-TB-KP solution retrieved so far, we update the values of variables BestHeuSol and BestHeuValue (lines 18–20). At lines 22–23 the counter j is incremented by one unit and the new problem $S^{(j)}$ is created (notice that it is safe to add the constraint (13) on $\hat{x}^{(j)}$ instead of $\hat{z}^{(j)}$, since solving the knapsack we have optimized the former into the latter, and there is no need to investigate further solutions containing $\hat{x}^{(j)}$). The best solution retrieved is returned at line 23, together with an indicator of the heuristic nature of the solution, in case this exit point is used. Notice that when the algorithm returns a heuristic solution, it is technically possible to post-calculate some upper bound, but we decided to omit this phase.

Analogously to what was seen in Section 4.1 for Algorithm 1, also for Algorithm 2 we will calculate a lower bound by solving a deterministic 01-KP to initialize variables and speed up the computation.

5. Experimental results

5.1. Benchmark instances

The benchmark set adopted for the experiments is that originally introduced in Monaci et al. (2022) for the 01-TB-KP and publicly available at Santini (2020). The set is composed of 600 instances with different characteristics, designed to match real-world needs, and to stress factors, in order to fairly compare solving methods with different characteristics. Five classes of instances are considered, the first four classes being based on the hard instances originally introduced in Pisinger (2005) for the 01-KP, and the last one being created to challenge the bounds discussed in Monaci et al. (2022). The instances have 100, 500, 1000 or 5000 items, with 10, 20 or 50% of the items being time-bombs. Considering the first four classes of instances, the time-bombs are either the items with the largest profits (classes 1 and 3) or those with the largest profit to weight ratio (classes 2 and 4). The explosion probabilities are either correlated to the profits (class 1), to the profit to weight ratios (class 2), or at random (classes 3 and 4). The instances of class 5 are instead artificially built to stress the solving methods: all the items have the same weight and the value price times probability of explosion is constant for all the items. Moreover, most of the items are time-bombs. We refer the interested reader to Monaci et al. (2022) for full details about the construction of all the instances.

5.2. Computational experiments

The algorithms discussed in Section 4 were implemented in Python, and Gurobi v11 (Gurobi Optimization, LLC, 2024) was used (with *MIPFocus*=3) to solve all the integer linear programs faced during the computation, while the deterministic knapsack problems were solved with the tools provided by the NetworkX package (Hagberg et al., 2008). All the experiments reported are based on a maximum computation time of 3600 s for each method/instance combination.

The methods proposed in Section 4 are compared in Table 1 with those available in the literature (and presented in Monaci et al., 2022). In detail, the non-linear model NL discussed in is attacked with both the solvers Baron (The Optimization Firm, 2024) and Couenne (COIN-OR, 2024), and the experiments were conducted on a computer equipped with an Intel Xeon E5630 processor running at 2.53 GHz (Santini, 2024) for these methods. Moreover, the Dynamic Programming (DP) and the Branch&Bound (B&B) exact approaches presented in Monaci et al. (2022) are considered, and for these methods a computer equipped with an Intel Xeon E5 processor running at 1.70 GHz (a more precise information is unfortunately unavailable) (Santini, 2024) was adopted. Finally the two algorithms discussed in Section 4 are run with different values of parameter ML in order to select the most promising settings. These last experiments were conducted on a computer equipped with an Apple M1 processor running at 2.06/3.23 GHz and with 8 GB of RAM. According to PassMark Software Pty Ltd (2024), our processor is approximately 3.7 times faster than the one used for non-linear solvers and between 1.1 and 3.0 times faster than the one used for DP and B&B. The results will therefore be affected by these differences in processor speed, but this will not affect the massive superiority of the new results in terms of speed.

For each method, the relevant indicators among the following ones are reported: *%gap* is the average optimality gap obtained at the end of the computation; *time* is the computation time in seconds required on average; *%opt* is the percentage of instances solved to optimality; *%valid* is the percentage of runs that were completed (the method Couenne suffers from numerical instability on these experiments); *nr iters* is the average number of iterations carried out. Notice that some entries are marked with a dash to indicate that the method under investigation was not able to provide solutions in the given time.

The results of Table 1 show that the new methods proposed in Section 4 are the only ones able to solve all the instances to optimality. Moreover, the computation time required is order of magnitudes lower than those of the previous methods, even when the different machines adopted for the experiments are taken into consideration. In terms of parameter tuning, it appears that the best performances are achieved with $ML = 1$ for both AlgorithmP and AlgorithmS, notwithstanding that the results appear very similar under all the settings. It can finally be observed that AlgorithmS normally requires slightly fewer iterations than AlgorithmP to prove optimality. In conclusion, if the target is to have high quality solutions, AlgorithmS is the best option, while in case upper bounds are of interest, AlgorithmP might be the answer, due to its capability of providing estimations from above.

In Table 2 an analysis on the performance of the most promising approaches is reported while aggregating the results by type of instance. The last line of the table contains statistics over all the instances.

Table 1
Experimental results by the exact methods available in the literature.

Method		Indicator	Size				
			100	500	1000	5000	All
NL – Baron (Monaci et al., 2022)		%gap	15.50	19.70	21.22	66.45	30.72
		Time	946.78	1372.78	1607.87	3461.52	1847.24
		%opt	74.67	64.00	59.33	10.00	52.00
NL – Couenne (Monaci et al., 2022)		%gap	7.84	19.33	20.25	78.67	31.89
		Time	1684.64	2565.82	2636.10	2976.03	2462.03
		%opt	54.36	32.89	30.60	17.33	33.85
		%valid	99.33	99.33	89.33	100.00	97.00
DP (Monaci et al., 2022)		Time	2642.58	–	–	–	2642.58
		%opt	45.33	–	–	–	45.33
B&B (Monaci et al., 2022)		%gap	0.00	0.06	0.29	0.36	0.18
		Time	5.49	109.55	163.16	266.88	136.27
		%opt	100.00	98.67	98.67	97.33	98.67
AlgorithmP	$ML = 1$	Time	12.70	1.77	1.24	1.94	4.41
		%opt	100.00	100.00	100.00	100.00	100.00
		nr iters	34.98	24.42	17.34	9.81	21.6
	$ML = 2$	Time	25.36	2.72	1.91	2.34	8.08
		%opt	99.33	100.00	100.00	100.00	99.83
		nr iters	38.42	24.38	17.34	9.94	22.52
	$ML = 3$	Time	25.92	3.70	2.56	2.58	8.69
		%opt	99.33	100.00	100.00	100.00	99.83
		nr iters	37.41	24.45	17.34	9.95	22.29
AlgorithmS	$ML = 1$	Time	0.44	1.18	1.56	1.52	1.18
		%opt	100.00	100.00	100.00	100.00	100.00
		nr iters	15.07	17.53	25.15	5.71	16.00
	$ML = 2$	Time	0.60	1.43	2.10	1.86	1.50
		%opt	100.00	100.00	100.00	100.00	100.00
		nr iters	15.08	17.53	24.93	6.07	15.91
	$ML = 3$	Time	0.74	1.65	2.63	2.26	1.82
		%opt	100.00	100.00	100.00	100.00	100.00
		nr iters	15.07	17.53	24.91	6.19	15.93

The new indicator *normalized time* is also added for method B&B. It contains the computation time normalized to the machine used in the present study, according to PassMark Software Pty Ltd (2024). Unfortunately an interval of times (in seconds) has to be provided due to the uncertainty about the processor originally used in Monaci et al. (2022) for the experiments (Santini, 2024).

Table 2 suggests the method B&B is the fastest one for a few of the smallest instances, but then the performance is worsening fast on the other instances. The new methods we propose are however not affected by the size of the instances, and present therefore a very promising scalability, differently from the method previously available in the literature. The results also suggest that the new methods have degradation of performance on some instances, without however a pattern on the characteristics emerges. This suggests that the methods might have some issue of robustness, which is although always recovered in a reasonable amount of computation time. When comparing the two new algorithms AlgorithmP and AlgorithmS, they seem to be equally prone to some degradation of performance in some cases, and in general the same conclusions can be drawn on the results of Table 1: AlgorithmS is in general faster (approximately twice as faster, on average), so it should be preferred. However, in case future application should generate instances harder to solve than those currently considered in the literature, AlgorithmP might become useful because it is the only one able to provide valid upper bounds, once it is impossible to prove optimality.

6. Conclusion

We presented two model-based algorithms for the 0–1 Time-bomb Knapsack Problem, both exploiting a well-known property of the logarithm operator to produce two different partial linearizations of a non-linear model describing the problem. An experimental comparison with the methods previously available in the literature, highlights the superiority of the new approaches, on top of benchmarking the two new model-based algorithms against each other. From the results it emerges that the new algorithms should be able to guarantee proven optimal solution for instances as large as practical applications require.

Future work will be about the application of model-based algorithms similar to those discussed in this paper for stochastic optimization problems sharing probabilistic patterns analogous to the 0–1 Time-bomb Knapsack Problem.

CRediT authorship contribution statement

Roberto Montemanni: Writing – review & editing, Writing – original draft, Visualization, Validation, Software, Resources, Methodology, Investigation, Formal analysis, Conceptualization. **Derek H. Smith:** Writing – review & editing, Writing – original draft, Validation, Investigation, Formal analysis.

Acknowledgments

The authors would like to thank the anonymous reviewers for the constructive suggestions provided.

Data availability

Data will be made available on request.

Table 2
Experimental comparison by type of instances.

Type	Size	Method									
		B&B (Monaci et al., 2022)				AlgorithmP ($ML = 1$)			AlgorithmS ($ML = 1$)		
		%gap	Time	Normalized time	%opt	Time	%opt	nr iters	Time	%opt	nr iters
Class 1	100	0.00	1.96	[0.65–1.78]	100.0	0.43	100.00	11.3	0.10	100.00	10.4
	500	0.00	11.73	[3.91–10.66]	100.0	0.73	100.00	9.8	0.36	100.00	12.6
	1000	0.00	7.91	[2.64–7.19]	100.0	0.28	100.00	10.1	0.36	100.00	11.0
	5000	0.01	197.58	[65.86–179.62]	96.7	0.85	100.00	5.6	0.80	100.00	4.0
	All	0.00	54.79	[18.26–49.81]	99.2	0.57	100.00	9.2	0.41	100.00	9.5
Class 2	100	0.00	5.15	[1.72–4.68]	100.0	1.16	100.00	19.6	1.36	100.00	18.4
	500	0.00	130.29	[43.43–118.45]	96.7	2.51	100.00	21.9	3.24	100.00	20.4
	1000	0.00	66.77	[22.26–60.70]	100.0	1.26	100.00	10.5	1.30	100.00	12.1
	5000	0.00	224.53	[74.84–204.12]	100.0	0.34	100.00	4.5	0.70	100.00	2.4
	All	0.00	106.68	[35.56–96.98]	99.2	1.31	100.00	14.1	1.65	100.00	13.3
Class 3	100	0.00	0.80	[0.27–0.73]	100.0	0.65	100.00	15.7	0.08	100.00	12.2
	500	0.00	10.84	[3.61–9.85]	100.0	1.14	100.00	26.5	0.85	100.00	27.0
	1000	0.71	143.09	[47.70–130.08]	96.7	1.26	100.00	22.7	1.54	100.00	27.4
	5000	0.15	236.96	[78.99–215.42]	96.7	6.97	100.00	25.3	4.08	100.00	13.7
	All	0.21	97.92	[32.64–89.02]	98.3	2.50	100.00	22.6	1.64	100.00	20.1
Class 4	100	0.00	3.48	[1.16–3.16]	100.0	61.07	100.00	89.0	0.41	100.00	11.7
	500	0.29	256.66	[85.55–233.33]	96.7	4.00	100.00	36.2	2.90	100.00	41.9
	1000	0.00	13.22	[4.41–12.02]	100.0	2.50	100.00	16.9	1.95	100.00	16.3
	5000	0.00	406.57	[135.52–369.61]	100.0	1.22	100.00	8.5	1.73	100.00	5.5
	All	0.07	169.98	[56.66–154.53]	99.2	17.19	100.00	37.7	1.75	100.00	18.8
Class 5	100	0.00	16.05	[5.35–14.59]	100.0	0.17	100.00	39.4	0.26	100.00	22.7
	500	0.00	138.23	[46.08–125.66]	100.0	0.47	100.00	27.6	0.48	100.00	23.8
	1000	0.77	584.82	[194.94—531.65]	96.7	0.91	100.00	26.5	0.74	100.00	20.8
	5000	1.66	268.75	[89.58–244.32]	93.3	0.32	100.00	5.1	0.29	100.00	3.1
	All	0.61	251.96	[83.99–229.05]	97.5	0.47	100.00	24.6	0.44	100.00	17.6
Overall		0.18	136.27	[45.42–123.88]	98.7	4.41	100.00	21.6	1.82	100.00	15.9

References

Bertsimas, D., Sim, M., 2003. The price of robustness. Oper. Res. (52), 35–53.

Cacchiani, V., Iori, M., Locatelli, A., Martello, S., 2022a. Knapsack problems — An overview of recent advances. Part I: Single knapsack problems. Comput. Oper. Res. 143, 105692.

Cacchiani, V., Iori, M., Locatelli, A., Martello, S., 2022b. Knapsack problems — An overview of recent advances. Part II: Multiple, multidimensional, and quadratic knapsack problems. Comput. Oper. Res. 143, 105693.

COIN-OR, 2024. Couenne solver. URL https://www.coin-or.org/Couenne/.

Dean, B., Goemans, M., Vondrák, J., 2008. Approximating the stochastic knapsack problem: The benefit of adaptivity. Math. Oper. Res. 4 (33), 945–964.

Donati, A.V., Montemanni, R., Gambardella, L.M., Rizzoli, A.E., 2003. Integration of a robust shortest path algorithm with a time dependent vehicle routing model and applications. In: Proceedings of the 3rd International Workshop on Scientific Use of Submarine Cables and Related Technologies. pp. 26–31.

Farrington, M., 2001. Safety of lithium batteries in transportation. J. Power Sources 1 (96), 260–265.

Galli, L., Martello, S., Rey, C., Toth, P., 2025a. The quadratic knapsack problem with setup. Comput. Oper. Res. 173, 106873.

Galli, L., Martello, S., Toth, P., 2025b. The quadratic knapsack problem. European J. Oper. Res. (in press).

Gurobi Optimization, LLC, 2024. Gurobi 11.0. URL https://www.gurobi.com.

Hagberg, A., Swart, P., Chult, D.S., 2008. Exploring Network Structure, Dynamics, and Function using NetworkX. Tech. Rep., Los Alamos National Lab. (LANL), Los Alamos, NM (United States), https://networkx.org/. (Accessed 14 March 2024).

Lisbona, D., Snee, T., 2011. A review of hazards associated with primary lithium and lithium–ion batteries. Process. Saf. Environ. Prot. 6 (89), 434–442.

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., Toth, P., 1990. Knapsack Problems: Algorithms and Computer Implementations. John Wiley & Sons, Inc., United States.

Merzifonluoglu, 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., 2003. On the robust knapsack problem. SIAM J. Optim. 4 (23), 1956–1982.

Monaci, M., Pike-Burke, C., Santini, A., 2022. Exact algorithms for the 0–1 time-bomb knapsack problem. Comput. Oper. Res. 145, 105848.

Montemanni, R., Leggieri, V., Triki, C., 2008. Mixed integer formulations for the probabilistic minimum energy broadcast problem in wireless networks. European J. Oper. Res. 190 (2), 578–585.

Morton, D., Wood, K., 1998. On a stochastic knapsack problem and generalizations. In: Advances in Computational and Stochastic Optimization, Logic Programming, and Heuristic Search. Springer, pp. 149–168.

PassMark Software Pty Ltd, 2024. CPU benchmarks. URL https://www.cpubenchmark.net/.

Pisinger, David, 2005. Where are the hard knapsack problems?. Comput. Oper. Res. 32 (9), 2271–2284.

Santini, A., 2020. The 0–1 time-bomb knapsack problem. URL https://github.com/alberto-santini/tbkp.

Santini, A., 2024. Private communication.

Srikantaiah, S., Kansal, A., Zhao, F., 2008. Energy-aware consolidation for cloud computing. In: Proceedings of the 2008 Usenix AnnualTechnical Conference.

Steinberg, E., Parks, M., 1979. A preference order dynamic program for a knapsack problem with stochastic rewards. J. Oper. Res. Soc. 2 (30), 141–147.

The Optimization Firm, 2024. Baron solver. URL https://minlp.com/baron-solver.