

# The min-Knapsack Problem with Compactness Constraints and Applications in Statistics

This preprint is available at <https://santini.in/>

A final version of this preprint is published as follows:

**Author 1:** Alberto Santini

**Author 2:** Enrico Malaguti

**Journal:** European Journal of Operational Research vol. 312(1), pp. 385–397 (2024).

The final version is available at:

<https://www.sciencedirect.com/science/article/pii/S0377221723005593>

You can cite the final version of this paper as:

```
@article{Santini2024,  
  title={The min-Knapsack Problem with Compactness Constraints and Applications in Statistics},  
  journal={European Journal of Operational Research},  
  author={Santini, Alberto and Malaguti, Enrico},  
  volume={312},  
  issue={1},  
  pages={385--397},  
  doi={10.1016/j.ejor.2023.07.020},  
  year={2024}  
}
```

# The min-Knapsack Problem with Compactness Constraints and Applications in Statistics

Alberto Santini<sup>1</sup> and Enrico Malaguti<sup>2</sup>

<sup>1</sup>Department of Economics and Business, Universitat Pompeu Fabra, Barcelona, Spain\*

<sup>2</sup>Dipartimento di Ingegneria dell'Energia Elettrica e dell'Informazione "Guglielmo Marconi",  
Università di Bologna, Italy<sup>†</sup>

19th August 2023

## Abstract

In the min-Knapsack problem, one is given a set of items, each having a certain cost and weight. The objective is to select a subset with minimum cost, such that the sum of the weights is not smaller than a given constant. In this paper we introduce an extension of the min-Knapsack problem with additional "compactness constraints" (mKPC), stating that selected items cannot lie too far apart from each other. This extension has applications in algorithms for change-point detection in time series and for variable selection in high-dimensional statistics. We propose three solution methods for the mKPC. The first two methods use the same Mixed-Integer Programming (MIP) formulation, but with two different approaches: either passing the complete model with a quadratic number of constraints to a black-box MIP solver or dynamically separating the constraints using a branch-and-cut algorithm. Numerical experiments highlight the advantages of this dynamic separation. The third approach is a dynamic programming labelling algorithm. Finally, we focus on the special case of the unit-cost mKPC (1c-mKPC), which has a specific interpretation in the context of the statistical applications mentioned above. We prove that the 1c-mKPC is solvable in polynomial time with a different ad-hoc dynamic programming algorithm. Experimental results show that this algorithm vastly outperforms both generic approaches for the mKPC, as well as a simple greedy heuristic from the literature.

**Keywords:** cutting; knapsack problems; applications in statistics; dynamic programming.

## 1 Introduction

In this paper, we present an extension of the *min-Knapsack* problem (Csirik et al. 1991) with applications in statistics, namely to change point detection in time series and to variable selection. Being an extension of min-Knapsack, the considered problem is  $\mathcal{NP}$ -complete. We also consider a special case of the problem which is both relevant for the statistical applications and solvable in polynomial time.

The min-Knapsack problem asks to select a subset of  $n$  items, each with weight  $w_j \geq 0$  and cost  $c_j \geq 0$  ( $j \in \{1, \dots, n\}$ ), such that the sum of the costs of the selected items is minimum, and their total weight is not smaller than a constant  $q \geq 0$ . We can formulate the min-Knapsack problem as the following integer programme, in which binary variable  $x_j$  takes value 1 iff the  $j$ -th item is selected:

$$\min \sum_{j=1}^n c_j x_j \tag{1}$$

$$\text{subject to } \sum_{j=1}^n w_j x_j \geq q \tag{2}$$

$$x_j \in \{0, 1\} \quad \forall j \in \{1, \dots, n\}. \tag{3}$$

---

\*alberto.santini@upf.edu

†enrico.malaguti@unibo.it

In this paper, we introduce a variant of the min-Knapsack problem, which we call the **min-Knapsack Problem with Compactness Constraints** (mKPC). Applications in time series analysis and high-dimensional statistics (see Section 1.1) motivate the study of this variant.

In the mKPC, there is a distance metric defined over the items. Consider two items  $i$  and  $j$  and assume, from now on and without loss of generality, that  $i \leq j$ . We define the distance between items as the difference of their indices, i.e.,  $j - i$ . We can think of the items as an ordered sequence, and we are interested in how far apart  $i$  and  $j$  lie in the sequence. With this notion of distance, we impose the additional condition that the set of selected items is compact. Formally, we consider a maximum distance parameter  $\Delta \in \mathbb{N}$ . If two items  $i$  and  $j$  are both selected and  $j - i > \Delta$ , then we require that there is at least another selected item between  $i$  and  $j$ . I.e., we require that there is a selected item  $k$  such that  $i < k < j$ .

Taking into account the above conditions, an IP formulation for the mKPC is the following:

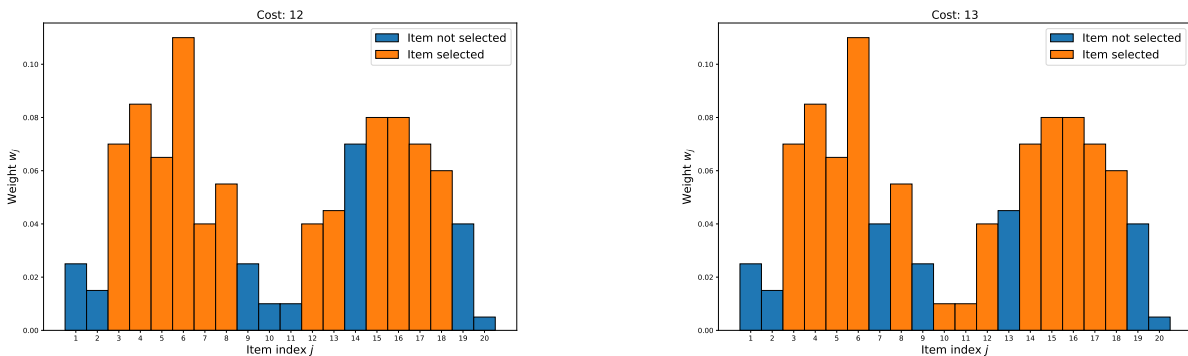
$$\min \sum_{j=1}^n c_j x_j \quad (4)$$

$$\text{subject to } \sum_{j=1}^n w_j x_j \geq q \quad (5)$$

$$x_i + x_j - 1 \leq \sum_{k=i+1}^{j-1} x_k \quad \forall i, j \in \{1, \dots, n\}, j > i + \Delta \quad (6)$$

$$x_j \in \{0, 1\} \quad \forall j \in \{1, \dots, n\}. \quad (7)$$

We denote constraints (6) the *compactness constraints*.



(a) Solution of the min-Knapsack problem.

(b) Solution of the mKPC.

Figure 1: Comparison of the solutions of the min-Knapsack problem and the mKPC on the same instance. Parameter  $\Delta = 2$ .

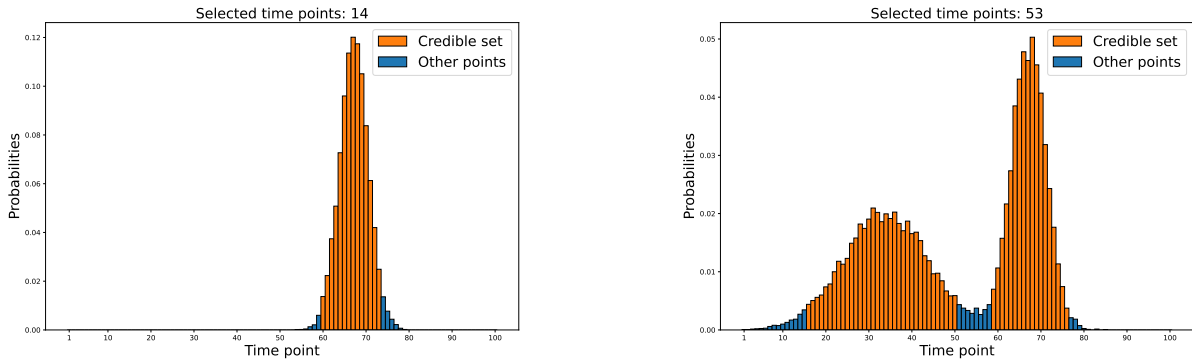
Figure 1 presents an example which shows the difference between the min-Knapsack problem (without compactness constraints), and the mKPC (with compactness constraints). Items lie on the  $x$  axis according to their index, and the bar heights indicate their weights. The value of parameter  $\Delta$  for the mKPC is set to 2 and  $c_j = 1$  for all items. An optimal solution of the min-Knapsack problem, depicted in Figure 1a, has total cost 12. However, it violates compactness constraints: items 8 and 12 (with distance  $4 > 2$ ) are both selected, but no other item between them is selected. Indeed, an optimal solution of the mKPC has a cost of 13, as shown in Figure 1b.

## 1.1 Applications in Statistics

The motivation for the mKPC comes from applications in statistics. In the following, we provide two examples: one from change-point detection in time series, and the other from variable selection in regression applied to fine-map genetic associations.

## Change-point detection in time series

Given a time series  $y_1, \dots, y_n$ , the objective of change-point detection is to identify whether the underlying probability distribution of  $y$  changes, how many times it does so, and at which time points. Cappello and Madrid Padilla (2022) introduced a state-of-the-art method, named PRISCA, for detecting changes in the variance of a Gaussian time series. They propose an iterative method which attempts to identify one change point at each iteration. To do so, at each iteration the authors build a *credible set* which quantifies the uncertainty associated with the presence of the change point. That is, they first build a discrete probability distribution over  $\{1, \dots, n\}$  which associates each time point with the probability that it is a change point. Next, they identify a level- $q$  credible set, i.e., a subset of  $\{1, \dots, n\}$  in which the sum of the probabilities is at least  $q$  (for a given threshold  $q \in [0, 1]$ ). For example, a level-0.95 credible set corresponds to a 95% probability that the set contains the change point.



(a) A possible credible set, when the change point is correctly identified.

(b) A possible credible set, when the change point is not correctly identified.

Figure 2: Two possible credible sets built during one iteration of the algorithm of Cappello and Madrid Padilla (2022) for a time-series with 100 time points.

Following a criterion of parsimony, it is desirable that the credible set contains as few elements as possible. Not all time points, however, must carry the same penalty if included in the credible set. For example, a time instant corresponding to an external shock might cost less in terms of parsimony compared to a time instant when no such shock occurred. Therefore, one can associate to each time point  $j$  a scaling factor  $c_j$  and minimise the sum of these factors. On the other hand, when no such information is present, one can just set  $c_j = 1$  for all time instants. As we will see in Section 1.3, using a unitary scaling factor decidedly simplifies the problem.

Another consideration regards the compactness of the credible sets: because each set should identify a single change point, its element should be “compact” and, ideally, distributed tightly around the real (unknown) change point. Figure 2a depicts this situation. The time points corresponding to the orange bars have a high chance of being change points. Because they are few and they concentrate around time point 67, a user can derive useful information about where the time series is likely to change. By contrast, Figure 2b shows a situation in which the algorithm failed to identify a single change point at a given iteration. Because Cappello and Madrid Padilla (2022, Sec. 3.2) declare the credible set produced at an iteration uninformative if it contains more than 50% of the points (using unitary scaling factors), the set depicted in Figure 2b would be discarded because it contains 53 out of 100 points.

Finally, note that once the value of parameter  $\Delta$  is fixed (usually to a small number such as 2 or 3), the problem of producing the most parsimonious credible set becomes our mKPC, in which the probability values associated to each time point take the role of the weights.

## Variable selection

Variable selection is an important problem in high-dimensional statistics. Consider the problem of selecting relevant independent variables in a regression model. In this case, variable selection can be used to improve the interpretability of the model by helping to learn which variables are *effect variables*. This is a typical task in bio-statistics, where one is more interested in learning which predictors affect a given outcome, rather than in building the model with the highest out-of-sample accuracy.

There are two important aspects which must be taken into account when performing variable selection. The first is that it is desirable to quantify the uncertainty about which variables should be selected. The second is that in real-world applications there often are multiple groups of variables which are highly correlated with each other. An interpretable variable selection method should inform the user about these groups and report its confidence in a given variable being selected.

Consider the example from Wang et al. (2020), who present a regression model with  $> 4$  independent variables in which variables  $X_1$  and  $X_4$  are effect variables, variable  $X_2$  is completely correlated with  $X_1$  and variable  $X_3$  is completely correlated with  $X_4$ . In this case, a helpful model would report that there are two effect variables: the first is one among  $X_1$  and  $X_2$ , and the second is one among  $X_3$  and  $X_4$ . Contrast this with many classical variable selection methods which would just arbitrarily pick one variable from each pair.

One way in which a model can create the desired report is via credible sets. In the example above, the helpful model could produce, for example, two level-0.95 credible sets  $\{1, 2\}$  and  $\{3, 4\}$ . SuSiE is a regression model introduced by Wang et al. (2020), with the objective of finding such credible sets. The algorithm used to fit SuSiE is iterative and aims at identifying one credible set at each iteration. To do so, it builds a discrete probability distribution over the index set  $\{1, \dots, n\}$  of independent variables. The probability value associated with index  $j$  represents the confidence that variable  $X_j$  belongs to the credible set. At each iteration, the algorithm selects a subset of indices such that the sum of the associated probabilities is larger or equal than a given user threshold  $q \in [0, 1]$ .

Similar to the time series application, a criterion of parsimony suggests that such a subset should contain as few variables as possible. Different from the previous application, however, it is not clear that there should be a “compactness” criterion or even that the independent variables can be ordered in a sequence. In general, one could define a distance function over independent variables such that strongly correlated ones lie close to each other and uncorrelated ones are far away from each other. In our example, however, we consider one specific but very important application area for the SuSiE model in which independent variables are naturally ordered: fine-map genetic associations. In the application presented by Wang et al. (2020), the independent variables represent loci in the human genome and the objective of the model is to understand how genetic variants affect cell splicing. The relevant property of the loci is that they allow a natural sequencing by considering them in the order in which they appear in the genome. The model should then try to produce credible sets containing nearby loci, as they are more likely to be part of the same gene. A too-wide set might instead indicate that the algorithm failed at identifying a credible set at the current iteration. Furthermore, scientists often have prior beliefs on which loci should influence cell splitting. They can encode these beliefs by applying a lower penalty  $c_j$  to loci which are likely to influence the outcome variable. These penalties are analogous to those presented in the time series application.

The problem of creating a compact credible set at each iteration can then be framed as a mKPC in which the probabilities that the independent variables belong to the set assume the role of the weights.

## 1.2 Related problems

In addition to applications in statistics discussed in the previous section, the mKPC has a specific combinatorial structure. As anticipated, the problem falls in the wide family of knapsack problems (see Kellerer, Pferschy and Pisinger 2004; Martello and Toth 1990). In particular, it extends the min-Knapsack problem by introducing compactness constraints. For the earliest results on the min-Knapsack

problem in English, we refer the reader to the seminal work of Csirik et al. (1991); for earlier works in Russian see, e.g., Babat (1975).

The special structure of compactness constraints can be represented by a graph  $G = (V, E)$  in which each item  $i$  corresponds to a vertex  $v_i \in V$ , and an edge  $\{v_i, v_j\} \in E$  is defined for each pair of vertices  $v_i$  and  $v_j$ ,  $i < j$ , such that  $j - i < \Delta$ . The mKPC asks to select a subset of  $V$  inducing a connected subgraph, such that the corresponding items optimise the associated min-Knapsack problem.

If instead of graph  $G$  we are given a generic graph, and if we also have to include a pre-defined subset  $T \subset V$  of vertices in the connected subgraph, the problem is known as the Connection Subgraph problem (see Conrad et al. 2007). This problem is strongly  $\mathcal{NP}$ -complete and remains so even when  $T = \emptyset$ . As discussed in Section 1.3, the mKPC (that is, the Connection Subgraph problem with  $T = \emptyset$  and the special structure of graph  $G$ ) remains  $\mathcal{NP}$ -complete.

We conclude by observing that the definition of the mKPC as a problem on a graph gives us an interpretation of inequalities (6) as a special case of the connectivity constraints introduced by Fischetti et al. (2017) to impose connectivity of Steiner trees.

### 1.3 Problem complexity

The mKPC is  $\mathcal{NP}$ -complete because it contains the min-Knapsack problem as a special case when  $\Delta = n$ . In the applications described above, however, it can often be the case that all items take unit cost (i.e.,  $c_j = 1$  for all  $i \in \{1, \dots, n\}$ ). This is the case, for example, when the user has no prior knowledge of which time instants of a time series are more likely to be change points, or which independent variables in a regression problem are more likely to be explanatory. We denote the special case of the mKPC with unit costs as 1c-mKPC. The following theorem establishes a strong result about the 1c-mKPC: namely, that it can be solved in polynomial time.

**Theorem 1.** *Consider the decision version of the 1c-mKPC: for a given integer number  $t \in \{1, \dots, n\}$ , we want to know whether there exists a feasible solution of the 1c-mKPC using at most  $t$  items. The decision version of the 1c-mKPC can be solved in polynomial time.*

*Proof.* Consider a Dynamic Programming (DP) table  $W$  with entries  $W(i, \ell)$  for each  $i \in \{1, \dots, n\}$  and  $\ell \in \{1, \dots, i\}$ . Entry  $W(i, \ell)$  will contain the maximum weight of a subset of  $\{1, \dots, i\}$  such that the set has size  $\ell$  and that the element of the set with the highest index is item  $i$ . This table can be trivially initialised with  $W(i, 1) = w_i$  for all  $i \in \{1, \dots, n\}$ . Furthermore, the following DP recursion is valid:

$$W(i, \ell) = \max_{j \in [i-\Delta], \dots, i-1} \{W(j, \ell-1)\} + w_i, \quad (8)$$

where notation  $[i - \Delta]$  is used as a shorthand for  $\max\{1, i - \Delta\}$ . Recursion (8) is valid because of the following observation. Any set of size  $\ell$  having item  $i$  as its highest-index element must contain at least one element in  $\{[i - \Delta], \dots, i - 1\}$  as its second-highest-index element. If that were not the case, in fact, the compactness constraint would be violated.

Finally, to know whether there is a subset of  $\{1, \dots, n\}$  of size at most  $t$  such that its elements have weight at least  $c$  and that satisfies compactness constraints, we must check that

$$\min\{\ell \in \{1, \dots, n\} \mid \exists i \in \{\ell, \dots, n\} \text{ s.t. } W(i, \ell) \geq c\} \leq t. \quad (9)$$

We now analyse the complexity of the above algorithm to conclude that it runs in polynomial time in the instance size  $n$ . Indeed, table  $W$  has size  $O(n^2)$  and we derive the worst-case complexity of computing an entry. To compute a generic entry  $W(i, \ell)$  through (8) we need to compare values in rows  $[i - \Delta], \dots, i - 1$  of column  $\ell - 1$ , i.e., we perform at most  $\Delta$  comparisons. Noting that the table can be built in increasing order of columns and rows (indeed,  $W$  is lower-triangular) and that  $\Delta \leq n$ , we conclude that the total complexity of the DP algorithm is  $O(n^3)$ .  $\square$

## 2 Solution approaches

In this section, we describe exact approaches for the mKPC. We also describe a greedy heuristic for the 1c-mKPC, used in the PRISCA package (Cappello 2022).

### 2.1 Integer Programming

The first approach consists in solving model (4)–(7) with a black-box integer programming solver. The model is compact because it uses  $O(n)$  variables and  $O(n^2)$  constraints.

**Strengthening compactness constraints.** Compactness constraints (6) state that if two items lying more than  $\Delta$  positions apart are selected, then at least another item between them must be selected. These constraints, however, can be made stronger. For example, if the two selected items lie at least  $2\Delta$  positions apart, then at least two further items between them shall also be selected. In general, (6) can be strengthened as follows:

$$\left\lfloor \frac{j-i-1}{\Delta} \right\rfloor (x_i + x_j - 1) \leq \sum_{k=i+1}^{j-1} x_k \quad \forall i, j \in \{1, \dots, n\}, j > i + \Delta. \quad (10)$$

The following example shows why these constraints help tighten the continuous relaxation of the mKPC. Consider an instance in which the two heaviest items are the first one and the last one: let  $n = 1002$ ,  $w_1 = w_{1002} = 0.495$ , and  $w_j = 10^{-4}$  for all other  $j \in \{2, \dots, 1001\}$ . Further assume that costs are all equal, that  $\Delta = 5$ , and that  $\alpha = 0.95$ . Without compactness constraints one might simply choose items 1 and 1002, obtaining a total weight of  $0.99 > 0.95$ . Due to compactness constraints, however, we must “link” these two items, taking other intermediate items. The most parsimonious way to achieve that is to take one every  $\Delta$  items, i.e., items 6, 11, ..., 1001. The optimal solution, therefore, selects  $2 + 200 = 202$  items.

When solving the continuous relaxation of the mKPC, however, an optimal solution is  $x_1 = x_{1002} = 1$ , and  $x_j = 10^{-3}$  for all other  $j \in \{2, \dots, 1001\}$ . Such a solution has cost 3 and does not violate any compactness constraint. For example, when  $i = 1$  and  $j = 1002$ , we have  $\sum_{k=i+1}^{j-1} x_k = 1000 \cdot 10^{-3} = 1$  and thus (6) is satisfied. On the other hand, the strengthened constraint (10) would be violated by such a solution:

$$\left\lfloor \frac{1001}{5} \right\rfloor (x_i + x_j - 1) = 200(1 + 1 - 1) = 200 \not\leq 1 = \sum_{k=i+1}^{j-1} x_k.$$

### 2.2 On-the-fly constraint generation

Formulation (4)–(7) has polynomial size, but the number of compactness constraints can be very large for large values of  $n$ . Their management can be impractical, and it can cause a degradation of black-box IP solvers’ performances, in particular during preprocessing and when solving linear programming relaxations. For this reason, we evaluate the effectiveness of a branch-and-cut approach in which we first remove the compactness constraints, and then generate them on-the-fly by separating infeasible integer and fractional solutions of the resulting relaxed problem. In the rest of this section, we derive the corresponding separation procedures.

**Integer solution separation.** The following procedure checks whether an integer solution  $x_1^*, \dots, x_n^*$  violates a compactness constraint. For each item  $i \in \{1, \dots, n\}$  with  $x_i^* = 1$ , we search the first item  $\sigma_i \in \{i + 1, \dots, n\}$  such that  $x_{\sigma_i}^* = 1$ . If  $\sigma_i > i + \Delta$ , then (6) is violated for  $j = \sigma_i$  and must be added to the formulation. Otherwise, there is no index  $j$  such that constraint (6) is violated for the index pair  $(i, j)$ . By stopping the algorithm after we find the first violated constraint (if any) would cut away the current infeasible integer solution. However, we can keep scanning items  $i$  even after we find one involved in the violation of a compactness constraint, thus attempting to separate other useful inequalities.

**Fractional solution separation.** Given a fractional solution  $x_1^*, \dots, x_n^*$ , the following procedure determines whether it violates a compactness constraint. For each item  $i \in \{1, \dots, n - \Delta - 1\}$  such that  $x_i^* > 0$ , let  $S = 0$ . Then:

1. For each item  $k \in \{i + 1, \dots, i + \Delta\}$ , update  $S$  with value  $S + x_k^*$ . If, at some point,  $S \geq 1$ , then there is no index  $j$  for which (6) is violated for the index pair  $(i, j)$ . We can then move to the next  $i$ .
2. Otherwise, start scanning items  $j \in \{i + \Delta + 1, \dots, n\}$  such that  $x_j^* > 0$ .
  - (a) If  $x_i + x_j - 1 > S$ , then the solution violates the compactness constraint for index pair  $(i, j)$ .
  - (b) Otherwise, update  $S$  with value  $S + x_j^*$  and move to the next  $j$ .

The validity of step 1 follows because condition  $S \geq 1$  makes the right-hand side of (6) larger or equal than 2 and, thus, the inequality holds. The condition in step 2.a corresponds exactly to a violation of (6), while step 2.b is needed to consider all items between  $i$  and  $j$ .

**Strengthened compactness constraints.** Finally, we observe that the separation procedure for compactness constraints can be modified in a straightforward way to detect and add violated inequalities (10) instead of the original (6). In particular, for the fractional case, it is enough to replace the condition in step 2.a with condition

$$\left\lfloor \frac{j - i - 1}{\Delta} \right\rfloor (x_i + x_j - 1) > S.$$

### 2.3 Dynamic Programming

In order to derive a DP algorithm for the (general) mKPC, we first introduce an auxiliary directed graph  $\mathcal{G} = (\mathcal{V}, \mathcal{A})$ . The vertex set contains a source node  $\sigma$ , a sink node  $\tau$ , and one node for each item. Overall,  $\mathcal{V} = \{\sigma, 1, \dots, n, \tau\}$ . The arc set  $\mathcal{A}$  contains:

- Arcs from  $\sigma$  to each node  $i \in \{1, \dots, n\}$ .
- Arcs from each node  $i \in \{1, \dots, n\}$  to  $\tau$ .
- An arc from node  $i$  to  $j$ , for each pair  $i, j \in \{1, \dots, n\}$  such that  $i < j \leq i + \Delta$ .

A feasible solution of the mKPC corresponds to a path in  $\mathcal{G}$  starting at  $\sigma$ , ending at  $\tau$ , and such that the weight collected at visited nodes is at least  $q$ .

To avoid the complete enumeration of all feasible solutions, we propose a labelling algorithm in which we associate a label to each partial path from  $\sigma$ . A label  $\mathcal{L} = (i, C, W)$  has three components: the last visited node  $i$ , the total cost  $C$  of visited nodes, and the total collected weight  $W$ . The initial label is  $\mathcal{L} = (\sigma, 0, 0)$ . Each time a label  $\mathcal{L} = (i, C, W)$  is extended from  $i$  to  $j$ , the new label  $\mathcal{L}' = (j, C', W')$  has components:

$$i' = j, \quad C' = \begin{cases} C + c_j & \text{if } j \in \{1, \dots, n\} \\ C & \text{if } j = \tau \end{cases}, \quad W' = \begin{cases} W + w_j & \text{if } j \in \{1, \dots, n\} \\ W & \text{if } j = \tau \end{cases}. \quad (11)$$

Optimal solutions of the mKPC correspond to labels such that  $i = \tau$ ,  $W \geq q$ , and  $C$  is minimal.

Note that, as soon as  $W \geq q$  for some label, the only sensible extension for that label is from the current node to the sink node  $\tau$ . Analogously, if  $W < q$ , then it does not make sense to extend that label to  $\tau$ , because the new label would correspond to an infeasible solution.

Consider two labels,  $\mathcal{L}_1 = (i, C_1, W_1)$  and  $\mathcal{L}_2 = (i, C_2, W_2)$ , referring to two partial paths to the same node  $i$ . If  $C_1 \leq C_2$  and  $W_1 \geq W_2$ , then no extension of  $\mathcal{L}_2$  up to the sink node  $\tau$  can correspond to a strictly better solution than the corresponding extension of  $\mathcal{L}_1$  along the same path. This observation leads to the following dominance rule:  $\mathcal{L}_1$  dominates  $\mathcal{L}_2$  if  $C_1 \leq C_2$ ,  $W_1 \geq W_2$ , and at least one of the



two inequalities is strict. In this case, one can discard label  $\mathcal{L}_2$ . In case both inequalities are actually equalities, one can discard either  $\mathcal{L}_1$  or  $\mathcal{L}_2$  (but not both), arbitrarily.

## 2.4 Greedy Heuristic for the 1c-mKPC

For the special case of the 1c-mKPC, we describe here the greedy heuristic procedure used in the PRISCA package (Cappello 2022) to determine whether a credible set corresponds to a valid change point. As mentioned in Section 1.1, the authors consider the case in which all costs are unitary, and they deem the credible set valid if their heuristic solution of the corresponding 1c-mKPC uses fewer than  $\frac{n}{2}$  items.

The greedy procedure aims at identifying a subset of items  $P \subseteq \{1, \dots, n\}$  with total weight at least  $q$  and satisfying the compactness constraints. The procedure starts by initialising  $P$  with a single item, namely the one with the highest weight:

$$P = \left\{ \operatorname{argmax}\{w_j \mid j \in \{1, \dots, n\}\} \right\}.$$

It then keeps augmenting  $P$  by adding, at each iteration, the heaviest item which is not yet selected and does not violate compactness constraints:

$$P \leftarrow P \cup \left\{ \operatorname{argmax}\{w_j \mid j \in \{1, \dots, n\} \setminus P, \exists i \in P : |j - i| \leq \Delta\} \right\}.$$

The algorithm stops as soon as  $\sum_{j \in P} w_j \geq q$ .

## 3 Computational results

In this section, we report the results of computational experiments to test the effectiveness of the algorithms presented in Section 2. The code was implemented in C++, using Gurobi 9.5 as the MIP solver. Experiments ran on a machine equipped with an Intel Xeon CPU running at 2.4 GHz and 4 GB RAM (increased to 8 GB for instances with  $n = 600$ ). The MIP solver was instructed to only use one thread. All algorithms used a time limit of 3600 s. The instances and the code used are available under an open-source licence (Santini 2022).

After describing the instance set used, we analyse the results of three sets of experiments:

1. Experiments to assess the impact of strengthened constraints (10).
2. Experiments to compare the compact formulation, the branch-and-cut algorithm, and the DP labelling algorithm for the mKPC.
3. Experiments to investigate the difficulty of solving the unit-cost version of the problem. To this end, on top of the above algorithms, we also add the DP algorithm for the 1c-mKPC (described in the proof of Theorem 1) and the greedy heuristic described in Section 2.4.

### 3.1 Instances

We consider two sets of instances. We obtained the first set, denoted S1, from the authors of (Cappello and Madrid Padilla 2022). This set consists of 300 instances with  $n \in \{40, 200\}$ ,  $q \in \{0.90, 0.95\}$ , and  $\Delta \in \{2, 3, 5\}$ . All costs are equal to 1 and, therefore, set S1 contains 1c-mKPC instances.

Because the costs in the S1 instances are all unitary, and the number of items is relatively low, we also generated a second set, denoted S2. This set contains 189 instances with  $n \in \{200, 400, 600\}$ ,  $q = 0.95$ , and  $\Delta \in \{2, 3, 5, 10\}$ . In the following, we explain how we generate the weights and the costs in the instances of set S2. We use three weight-generation methods:

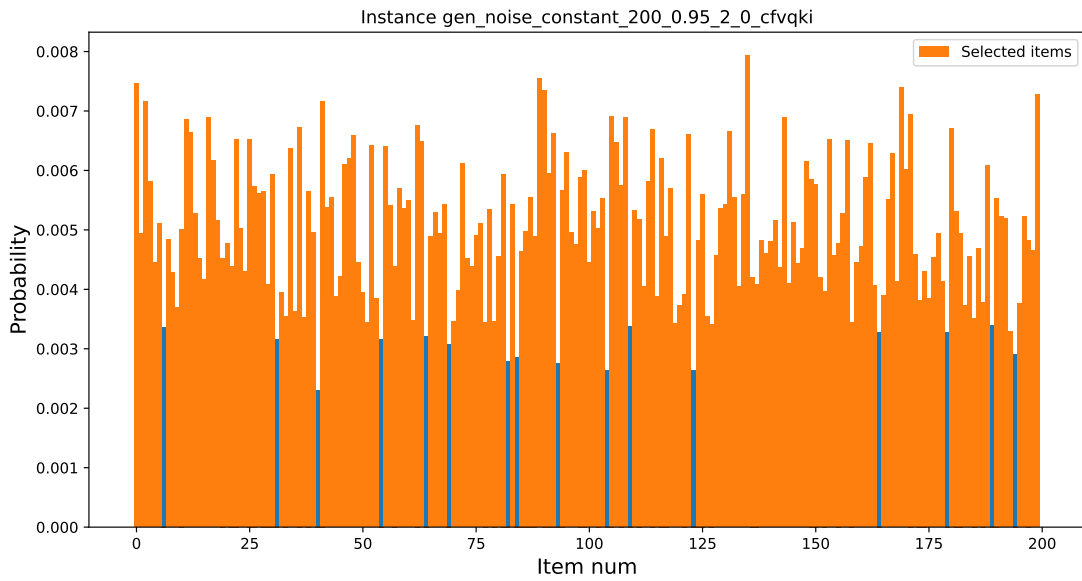


Figure 3: Example NOISE instance with its optimal solution.

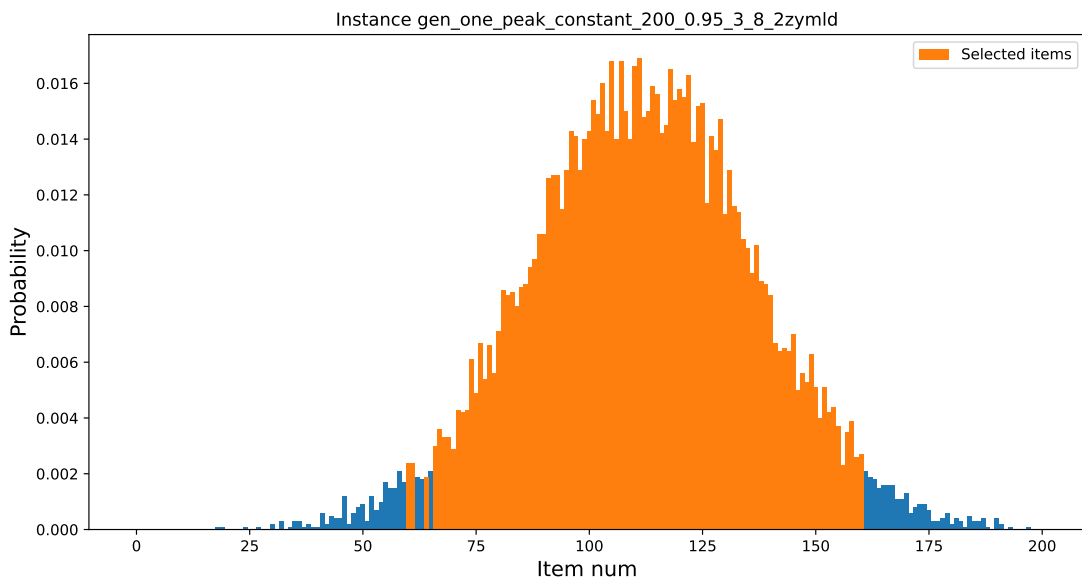


Figure 4: Example ONEPEAK instance with its optimal solution.

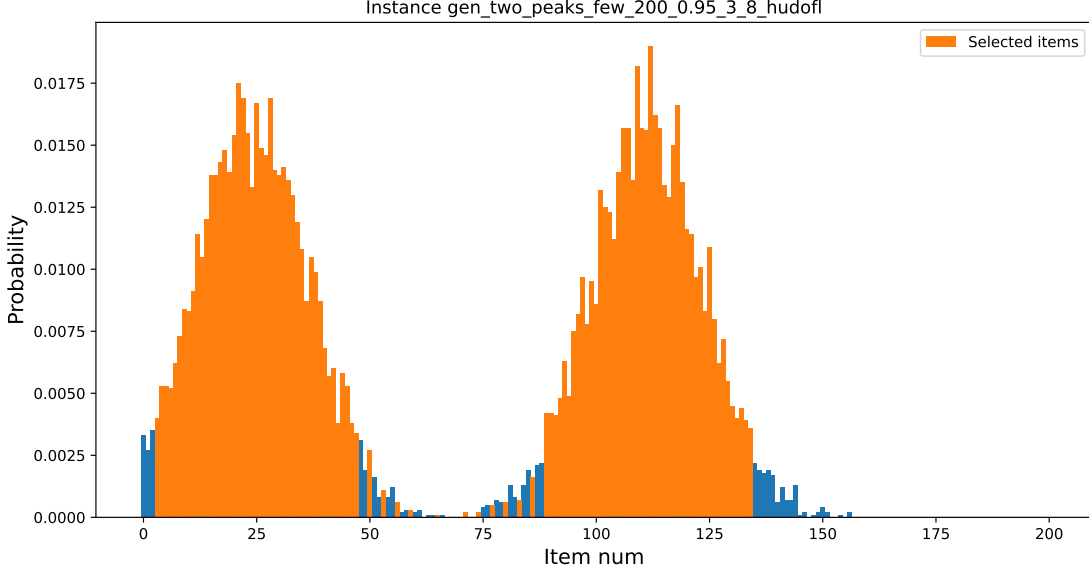


Figure 5: Example TWOPEAKS instance with its optimal solution.

- The NOISE method first assigns each item  $j$  a weight

$$w'_j = \frac{1}{n} + \mathcal{N}\left(0, \frac{1}{4n}\right),$$

where  $\mathcal{N}(\lambda, \sigma)$  denotes a normal distribution with location  $\lambda$  and scale  $\sigma$ . To avoid numerical issues, we also ensure that no weight is smaller than  $10^{-12}$ , i.e., we set

$$w'_j \leftarrow \max\{w'_j, 10^{-12}\}.$$

Because the sum of the above weights is not necessarily equal to one, we finally normalise them:

$$w_j = \frac{w'_j}{\sum_{i=1}^n w'_i} \quad \forall j \in \{1, \dots, n\}. \quad (12)$$

Figure 3 shows an example of a NOISE instance, with its optimal solution represented in orange. The  $y$  axis, labelled “Probability” refers to the statistical application mentioned in Section 1.1, in which item weights represent probabilities. NOISE instances tend to require a large fraction of selected items to reach the target weight of  $q = 0.95$ .

- The ONEPEAK method proceeds as follows. It first chooses a random location  $\lambda$  between 1 and  $n$ , sampling from a truncated normal distribution with location  $\frac{n}{2}$  and scale  $\frac{n}{4}$ , and rounding to the nearest integer. It then generates an instance in which the weights have a peak around  $\lambda$ , i.e., an instance similar to the one depicted in Figure 2a. To this end, it considers another truncated normal distribution between 1 and  $n$ , with location  $\lambda$  and scale  $\frac{n}{k}$ . Here  $k \in \{8, 16, 32\}$  is an instance generation parameter. Weights will be more tightly distributed around the peak when  $k$  is larger. The method samples 5000 times from this distribution and builds the corresponding histogram with  $n$  bars. The  $j$ -th bar counts how many samples fell in interval  $[j, j+1)$ . The weight  $w'_j$  of the  $j$ -th item is then set as the height of the  $j$ -th bar of the histogram. Finally, weights  $w_j$  are obtained by normalisation as in eq. (12). Figure 4 shows an example of a ONEPEAK instance.
- The TWOPEAKS method is similar to ONEPEAK, except that the histogram is built by sampling from the sum of two truncated normal distributions with locations  $\lambda_1$  and  $\lambda_2$ , and common scale  $\frac{n}{2k}$ . Intuitively,  $\lambda_1$  and  $\lambda_2$  are the locations of two peaks, and we produce an instance in which weights look similar to those presented in Figure 2b. The values of the two locations are drawn from two further truncated normal distributions between 1 and  $n$ , and rounded to the nearest

integer. The first distribution has location  $\frac{n}{3}$ , the second one has location  $\frac{2n}{3}$ , and both have scale  $\frac{n}{6}$ . Figure 5 shows an example of a TWOPEAKS instance.

We use three costs generation methods:

- The CONSTANT method simply assigns unit costs to all items and allows us to extend the results obtained on the S1 set to larger instances with different weight types.
- The FEW method aims at modelling real-life statistical applications, in which few items have a small cost, and all other items have a constant larger one. In particular, it first selects  $\frac{n}{100}$  items using a roulette wheel method with probabilities equal to item weights. It then assigns these items a weight of 0.10, and all other items a weight of 1. The reason we use roulette wheel selection is that, in the application, the items with the lower costs correspond to time instants with a higher prior probability of containing a change point. These items are thus also more likely to be detected by the algorithm and, as a consequence, to have a larger weight. Therefore, assuming that the prior knowledge is accurate and that the algorithm works correctly, items with larger weights are more likely to have lower costs.
- The RANDOM method assigns each item a cost uniformly distributed in the interval  $[1, 10]$ .

Note that we have three possible values for parameter  $n$ , three values for parameter  $\Delta$ , and three for the cost generation method. Their combination gives 27 instances built using weight generation method NOISE. Furthermore, for each of these 27 combinations, we have 3 possible values for parameter  $k$ , yielding 81 instances for each of the ONEPEAK and TWOPEAKS weight generation methods. Overall, we then construct  $27 + 2 \times 81 = 189$  instances.

### 3.2 Computational experiments

In this section, we present the results of computational experiments on the instances described in Section 3.1. We first investigate the role of strengthened inequalities (10) on the compact formulation and the branch-and-cut (B&C) algorithm. Next, we compare these two algorithms with the labelling algorithm introduced in Section 2.3. We present the results of these comparisons using instances of set S2 because these are larger and more varied. Finally, we compare our approaches (including the DP one introduced via Theorem 1) with the greedy heuristic of the PRISCA package (Cappello 2022) on 1c-mKPC instances. This comparison allows us to assess the advantage given by exact algorithms over a heuristic one, on instances relevant to data scientists.

**Impact of strengthened compactness constraints.** We use two relevant metrics to assess the impact of strengthened inequalities (10):

1. The percentage optimality gap, i.e., the gap between the best primal and dual bounds returned by each algorithm within the time limit. This metric is denoted as GAP% and is defined as follows:

$$\text{GAP}\% = 100 \cdot \frac{\text{UB} - \text{LB}}{\text{UB}},$$

where “UB” indicates the best primal solution and LB is the tightest dual bound returned by the solver. GAP% corresponds to the familiar gap returned by black-box integer programming solvers and depends on both the quality of the primal and dual bound.

2. The second metric is the solution time in seconds, including the time spent creating the model and exploring the branch-and-bound tree. It is denoted by TIME (s).

We also note that instances generated using weight types TWOPEAKS are considerably harder than the other instances. Therefore, we present the results obtained on NOISE and ONEPEAK instances separately from those obtained on TWOPEAKS instances. After commenting on these results, we will come back to the difficulty of TWOPEAKS instances, and we will explain what sets them apart from the other instances.

$n$	Weights	Costs	Compact MIP		B&C	
			with (6)	with (10)	with (6)	with (10)
			TIME (s)	TIME (s)	TIME (s)	TIME (s)
200	NOISE	CONSTANT	3.30	5.26	0.00	0.01
		FEW	2.87	4.69	0.01	0.00
		RANDOM	3.93	8.12	0.01	0.01
	ONEPEAK	CONSTANT	5.16	4.21	0.00	0.01
		FEW	3.47	4.91	0.00	0.00
		RANDOM	5.72	6.07	0.08	0.04
400	NOISE	CONSTANT	40.04	32.24	0.00	0.01
		FEW	44.41	133.46	0.01	0.01
		RANDOM	44.76	77.12	0.06	0.05
	ONEPEAK	CONSTANT	62.42	84.51	0.00	0.00
		FEW	67.64	78.56	0.02	0.02
		RANDOM	77.67	153.64	0.13	0.04
600	NOISE	CONSTANT	243.71	223.85	0.01	0.01
		FEW	269.98	325.08	0.01	0.01
		RANDOM	288.32	207.23	0.04	0.02
	ONEPEAK	CONSTANT	774.32	1167.89	0.01	0.00
		FEW	586.40	556.84	0.04	0.03
		RANDOM	518.15	663.14	0.35	0.08
Overall			201.23	254.90	0.06	0.02

Table 1: Impact of strengthened inequalities (10) on the performance of the Compact Formulation and the Branch & Cut algorithm. The table refers to instances whose weights are generated with the NOISE and ONEPEAK methods.

$n$	Costs	Compact MIP		Compact MIP		B&C		B&C	
		with (6)		with (10)		with (6)		with (10)	
		GAP%	TIME (s)	GAP%	TIME (s)	GAP%	TIME (s)	GAP%	TIME (s)
200	CONSTANT	1.01	441.22	0.27	787.49	11.04	1615.14	0.71	812.96
	FEW	3.17	862.32	0.02	706.92	10.49	2477.93	0.40	560.32
	RANDOM	5.38	407.93	0.00	209.74	1.59	875.76	0.39	402.81
400	CONSTANT	8.90	1656.18	2.42	1282.71	15.15	2400.00	2.11	1630.98
	FEW	24.62	2857.35	4.60	3088.66	20.09	2823.07	8.32	2801.44
	RANDOM	6.81	1473.01	5.42	2329.48	19.93	2800.02	4.25	1760.49
600	CONSTANT	19.86	2153.77	6.56	2151.10	19.56	2000.03	16.59	1629.75
	FEW	20.70	2830.83	7.72	3006.47	10.63	2950.59	5.98	2409.40
	RANDOM	19.03	2111.40	7.31	2629.66	21.43	2400.55	6.99	2008.91
Overall		12.16	1643.78	3.81	1799.14	14.36	2071.66	5.08	1557.45

Table 2: Impact of strengthened inequalities (10) on the performance of the Compact Formulation and the Branch & Cut algorithm. The table refers to instances whose weights are generated with the TWOPEAKS method.

Table 1 reports the results on the NOISE and ONEPEAK instances of set S2. Because all algorithms solve to optimality all instances with up to 600 items, Table 1 is only reporting the runtimes. Note how the runtimes are very different for the complete compact formulation and for the B&C algorithm. For the largest instances, for example, the average runtimes needed to solve the compact formulation are in the order of hundreds of seconds. The B&C algorithm, on the other hand, closes these instances in a few hundredths of a second: a difference of five orders of magnitude. Regarding the effect of strengthened compactness constraints, we note that they do not seem to help when solving the full compact formulation. If anything, in fact, they slightly increase the computation time. On the other hand, they reduce the computation time of the B&C algorithm.

Table 2 presents the results on the TWOPEAKS instances. These instances are considerably harder to solve: in several cases, the solvers run out of time without solving the model to optimality. Even when they solve the model to optimality, it takes on average much longer compared with NOISE and ONEPEAK instances. For these instances, the strengthening constraints have a considerable effect on the solvers. Indeed, by using the strengthened inequalities (10) the average gaps are roughly reduced by two thirds. We also observe that, on these harder instances, the B&C algorithm loses its advantage on the compact formulation. The gaps produced by B&C are slightly worse, while the runtimes are comparable.

**Peculiarity of the TwoPeaks instances.** As Tables 1 and 2 show, TWOPEAKS instances are much harder to solve using branch-and-bound methods, compared with the other instances. The reason lies in the characteristics of the optimal solution of the continuous relaxation of the mKPC. Solutions of TWOPEAKS instances have a large number of fractional items and the value of the corresponding variables  $x$  are closer to 0.5. This implies that much more branching is necessary while exploring the branch-and-bound tree. To appreciate the extent by which TWOPEAKS instances differ from the other instances, Figure 6 shows boxplots of two metrics relative to the optimal solution of the continuous relaxation of the mKPC, divided by weight generation method. Let  $x_1^*, \dots, x_n^*$  be such a solution. Metric FRAC% gives the percentage of items selected fractionally in the solution, i.e.,

$$\text{FRAC}\% = 100 \cdot \frac{\left| \{j \in \{1, \dots, n\} : 0 < x_j^* < 1\} \right|}{n}.$$

Metric FRACGINI is the normalised Gini coefficient of the fractional variables, i.e.,

$$\text{FRACGINI} = \frac{\sum_{j=1}^n x_j^*(1 - x_j^*)}{\left| \{j \in \{1, \dots, n\} : 0 < x_j^* < 1\} \right|}.$$

The value of this metric is higher when many  $x_j^*$  are concentrated around 0.5, while it is lower when the  $x_j^*$  take values close to 0 or 1. Values  $x_j^* \in \{0, 1\}$  do not contribute to the sum at the numerator. Therefore, solutions with more fractional items have more non-zero terms in the sum at the numerator. To compensate for this fact we normalise dividing by the number of fractional items.

**Comparison of the algorithms for the mKPC.** Table 3 compares the performance of three approaches for the mKPC. Because strengthened inequalities (10) result in lower gaps, we enable them for both the branch-and-cut algorithm and the compact formulation.

Table 3 reports the following metrics:

1. OPT% is the percentage of instances in each row for which the algorithm found a provably optimal solution.
2. PGAP% is the percentage primal gap, i.e., the gap between the best primal solution found by each algorithm and the best known primal solution. We use PGAP% instead of GAP% because the labelling algorithm does not provide any dual bound when it cannot solve an instance within the time limit.

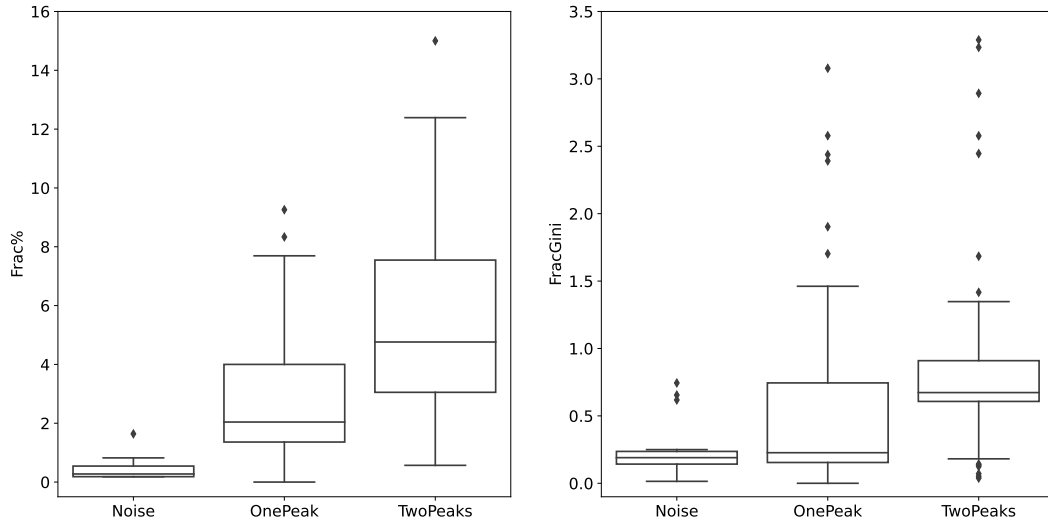


Figure 6: Left: percentage of items selected fractionally in the optimal solution of the continuous relaxation of the mKPC. Right: normalised Gini coefficient showing how close the fractional values are to 0.5 (the higher the value, the closer to 0.5).

$n$	Weights	Costs	B&C with (10)			Compact MIP with (10)			Labelling			
			OPT%	PGAP%	TIME (s)	OPT%	PGAP%	TIME (s)	FEAS%	OPT%	PGAP%	TIME (s)
200	NOISE	CONSTANT	100.00	0.00	0.01	100.00	0.00	5.26	100.00	100.00	0.00	1.74
		FEW	100.00	0.00	0.00	100.00	0.00	4.69	100.00	100.00	0.00	3.40
		RANDOM	100.00	0.00	0.01	100.00	0.00	8.12	66.67	33.33	67.89	2599.24
	ONEPEAK	CONSTANT	100.00	0.00	0.01	100.00	0.00	4.21	100.00	100.00	0.00	1.52
		FEW	100.00	0.00	0.00	100.00	0.00	4.91	100.00	100.00	0.00	1.75
		RANDOM	100.00	0.00	0.04	100.00	0.00	6.07	100.00	88.89	0.05	1141.71
	TWOPEAKS	CONSTANT	77.78	0.36	812.96	88.89	0.14	787.49	100.00	100.00	0.00	1.29
		FEW	88.89	0.23	560.32	88.89	0.00	706.92	100.00	100.00	0.00	1.25
		RANDOM	88.89	0.00	402.81	100.00	0.00	209.74	100.00	100.00	0.00	58.83
400	NOISE	CONSTANT	100.00	0.00	0.01	100.00	0.00	32.24	100.00	100.00	0.00	31.48
		FEW	100.00	0.00	0.01	100.00	0.00	133.46	100.00	100.00	0.00	90.07
		RANDOM	100.00	0.00	0.05	100.00	0.00	77.12	0.00	0.00	100.00	3600.00
	ONEPEAK	CONSTANT	100.00	0.00	0.00	100.00	0.00	84.51	100.00	100.00	0.00	17.86
		FEW	100.00	0.00	0.02	100.00	0.00	78.56	100.00	100.00	0.00	27.01
		RANDOM	100.00	0.00	0.04	100.00	0.00	153.64	100.00	88.89	0.01	2055.61
	TWOPEAKS	CONSTANT	66.67	1.91	1630.98	77.78	1.86	1282.71	100.00	100.00	0.00	20.87
		FEW	22.22	5.71	2801.44	22.22	3.52	3088.66	100.00	100.00	0.00	37.88
		RANDOM	55.56	2.71	1760.49	44.44	2.70	2329.48	88.89	66.67	13.42	2118.83
600	NOISE	CONSTANT	100.00	0.00	0.01	100.00	0.00	223.85	100.00	100.00	0.00	213.96
		FEW	100.00	0.00	0.01	100.00	0.00	325.08	100.00	100.00	0.00	816.55
		RANDOM	100.00	0.00	0.02	100.00	0.00	207.23	0.00	0.00	100.00	3600.00
	ONEPEAK	CONSTANT	100.00	0.00	0.00	100.00	0.00	1167.89	100.00	88.89	0.07	493.86
		FEW	100.00	0.00	0.03	100.00	0.00	556.84	100.00	100.00	0.00	293.43
		RANDOM	100.00	0.00	0.08	100.00	0.00	663.14	66.67	44.44	33.38	3009.45
	TWOPEAKS	CONSTANT	55.56	13.61	1629.75	44.44	5.11	2151.10	100.00	100.00	0.00	229.95
		FEW	33.33	4.78	2409.40	22.22	4.71	3006.47	100.00	77.78	24.13	424.97
		RANDOM	44.44	5.60	2008.91	33.33	6.02	2629.66	44.44	11.11	59.17	3506.86
Overall			82.54	1.66	667.49	82.01	1.15	916.71	91.53	84.66	10.45	699.87

Table 3: Comparison of the MIP-based approaches (the branch-and-cut and the compact formulation) with the labelling algorithm presented in Section 2.3, on the S2 instances.

3. Finally, we observe that the labelling algorithm can terminate in three different states. If it completes before the time limit, it has found the optimal solution. If it times out and there is at least one label extended to the sink node  $\tau$ , then the algorithm can be used as a heuristic: any label extended to the sink node corresponds to a feasible solution. The algorithm can return the best such solution, although it cannot prove or disprove its optimality while there are still unextended labels. If the algorithm times out and no label was extended to  $\tau$ , then we do not even have a feasible solution to compute  $\text{PGAP}\%$ . Therefore, we introduce the additional column  $\text{FEAS}\%$  for the labelling algorithm. It contains the percentage of instances in each row for which the algorithm found at least one feasible solution.

To ensure a fair comparison, we compute averages using a  $\text{PGAP}\%$  of 100 % and a  $\text{TIME (s)}$  of 3600s when the labelling algorithm does not give any feasible solution.

Table 3 shows that branch-and-cut is usually able to find more optima than the other algorithms and in a shorter time. The average primal gap, however, is slightly lower for the compact MIP formulation. The labelling algorithm does not always manage to produce a feasible solution. In particular, its performance degrades for instances with cost type `RANDOM` and, to a lesser extent, with weight type `NOISE`. For these instances, in fact, dominance is less likely and the labelling algorithm becomes similar to a complete enumeration of feasible solutions.

When the labelling algorithm finds feasible solutions, however, they are often optimal. In these cases, branch-and-cut usually also finds optimal solutions, and in a shorter time. A notable exception are `TWOPEAKS` instances: when costs are `CONSTANT` or `FEW`, labelling is the best-performing algorithm, finding more optima in a considerably shorter time.

**Experiments on 1c-mKPC instances.** Finally, we report the performances of five algorithms on the unit-cost mKPC. In addition to the three algorithms considered above, we add the DP approach presented in the proof of Theorem 1 and the greedy heuristic introduced in Section 2.4. The test set for this experiment includes both `S1` instances, and `S2` instances with `CONSTANT` costs.

The results show that the tailored DP approach vastly outperforms all other algorithms. It solves all instances, even the largest ones with  $n = 600$ , in less than two-thousandths of a second per instance. The greedy heuristic is even faster (all measured times were under 0.000 05 s) but often fails at identifying the optimal solution, especially when the size of the instance grows. We can conclude that specialised approaches for the 1c-mKPC are well justified and that there is no reason to use heuristic algorithms because the exact DP approach we propose is extremely fast in practice.

## 4 Conclusions

In this paper, we introduced the min-Knapsack Problem with Compactness Constraints (mKPC), an extension of the classical min-Knapsack problem. The motivation for studying the mKPC is that it arises as a sub-problem in two state-of-the-art algorithms recently introduced in the statistical community. These are the `PRISCA` algorithm of Cappello and Madrid Padilla (2022) for detection of change points in time series, and the `SuSiE` algorithm of Wang et al. (2020) for variable selection in high-dimensional regression.

We proposed three approaches to solve the mKPC: a compact formulation with a quadratic number of constraints, a branch-and-cut approach in which these constraints are separated dynamically, and a labelling algorithm. Despite branch-and-cut being more often used when the number of constraints is exponential in the problem size, computational experiments proved that this approach can also be useful when dealing with compact models. In particular, the branch-and-cut algorithm solves the largest number of instances to optimality and is orders of magnitude faster than solving the entire formulation with the state-of-the-art black-box solver `Gurobi`. Computational experiments also showed that the difficulty of the problem depends considerably on instance characteristics. In particular, instances with a particular double-peak structure are harder to solve to optimality.



Algorithm	$n$	OPT%	PGAP%	TIME (s)
B&C with (10)	40	100.00	0.00	0.0010
	200	99.23	0.01	28.9298
	400	76.19	0.36	698.9947
	600	76.19	5.22	698.4682
Compact MIP with (10)	40	100.00	0.00	0.0190
	200	99.62	0.00	40.8350
	400	80.95	0.33	590.5545
	600	71.43	1.07	1454.4012
Labelling (Section 2.3)	40	100.00	0.00	0.0007
	200	100.00	0.00	0.6564
	400	100.00	0.00	21.0971
	600	100.00	0.00	212.1984
Dynamic Programming (Theorem 1)	40	100.00	0.00	0.0000
	200	100.00	0.00	0.0000
	400	100.00	0.00	0.0005
	600	100.00	0.00	0.0021
Greedy (Section 2.4)	40	96.67	0.30	0.0000
	200	87.74	1.36	0.0000
	400	66.67	10.73	0.0000
	600	61.90	10.54	0.0000

Table 4: Comparison of four algorithms on the unit-cost instances of sets S1 and S2.

Finally, we focussed our attention on a special case of the mKPC, named the unit-cost mKPC (1c-mKPC). This problem is especially relevant for the statistical applications because it corresponds to the case in which the user of the PRISCA and SuSiE algorithms mentioned above has no prior knowledge of, respectively, which time instants and which features are more likely to be selected. We proved that the 1c-mKPC is solvable in polynomial time, and we proposed a specific dynamic programming algorithm. Computational results clearly show that using this algorithm is better than both the generic mKPC approaches and a greedy heuristic from the statistics literature.

## Acknowledgements

We are grateful to Lorenzo Cappello and Oscar Madrid Padilla for sharing with us the instances of set S1, and to Lorenzo Cappello for fruitful discussions on this problem. This research did not receive any specific grant from funding agencies in the public, commercial, or not-for-profit sectors.

## References

- Babat, L. (1975). ‘Linear functionals on the  $n$ -dimensional unit cube’. In: *Reports of the Academy of Sciences of the Soviet Union* 221 (4), pp. 761–762.
- Cappello, L. (2022). *prisca*. GitHub repository. URL: <https://github.com/lorenzocapp/prisca>.
- Cappello, L. and O. H. Madrid Padilla (2022). *Variance change point detection with credible sets*. arXiv: 2211.14097 [stat]. URL: <https://arxiv.org/abs/2211.14097>.
- Conrad, J., C. Gomes, W.-J. van Hoeve, A. Sabharwal and J. Suter (2007). ‘Connections in Networks: Hardness of Feasibility Versus Optimality’. In: *International Conference on Integration of Artificial Intelligence (AI) and Operations Research (OR) Techniques in Constraint Programming* (23rd–26th May 2007). Ed. by P. Van Hentenryck and L. Wolsey. Brussels, Belgium: Springer, pp. 16–28. DOI: 10.1007/978-3-540-72397-4\_2.

- Csirik, J., H. Frenk, M. Labbé and S. Zhang (1991). ‘Heuristics for the 0–1 min-Knapsack Problem’. In: *Acta Cybernetica* 10.1–2, pp. 15–20.
- Fischetti, M., M. Leitner, I. Ljubic, M. Luipersbeck, M. Monaci, M. Resch, D. Salvagnin and M. Sinnl (2017). ‘Thinning out Steiner trees: a node-based model for uniform edge costs’. In: *Mathematical Programming Computation* 9, pp. 203–229. DOI: 10.1007/s12532-016-0111-0.
- Kellerer, H., U. Pferschy and D. Pisinger (2004). *Knapsack Problems*. Springer. ISBN: 9783642073113.
- Martello, S. and P. Toth (1990). *Knapsack Problems. Algorithms and computer implementations*. Wiley. ISBN: 9780471924203.
- Santini, A. (29th Dec. 2022). *Algorithms for the min-Knapsack problem with compactness constraints*. GitHub repository. DOI: 10.5281/zenodo.7492799. URL: <https://github.com/alberto-santini/min-knapsack-with-compactness>.
- Wang, G., A. Sarkar, P. Carbonetto and M. Stephens (2020). ‘A simple new approach to variable selection in regression, with application to genetic fine mapping’. In: *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 82 (5), pp. 1273–1300. DOI: 10.1111/rssb.12388.