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

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

Alberto Santini[1] and Enrico Malaguti[2]

[1]Department of Economics and Business, Universitat Pompeu Fabra, Barcelona, Spain*
[2]Dipartimento di Ingegneria dell'Energia Elettrica e dell'Informazione "Guglielmo Marconi", Università di Bologna, Spain†

18th May 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 statistics, including in algorithms for change-point detection in time series. 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, including to change point detection in time series. 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.

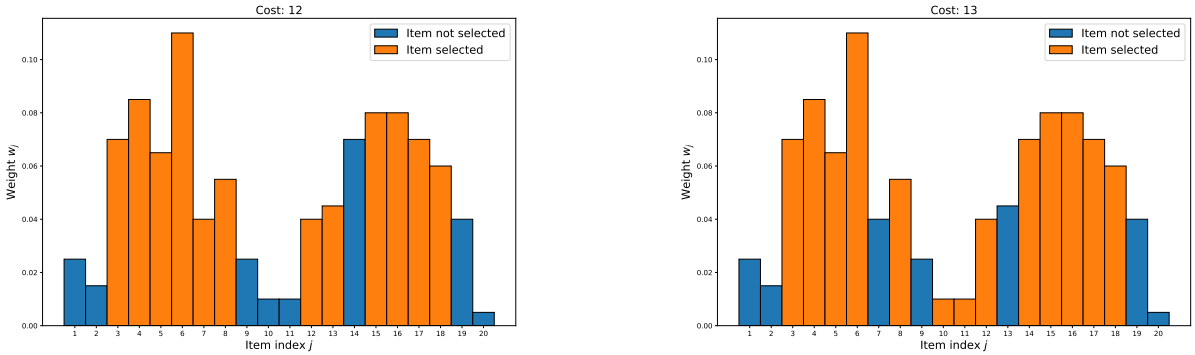
The min-Knapsack problem asks to select a subset of n items, each with weight $w_j \geq 0$ and cost $c_j \geq 0$ ($j \in \{1, \ldots, n\}$), such that the sum of the costs of the selected items is minimum, and their total weight is not smaller than a constant $q \geq 0$.

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 $i < j$. 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

*alberto.santini@upf.edu
†enrico.malaguti@unibo.it

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 $i < k < j$.



(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 Δ 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 Motivation

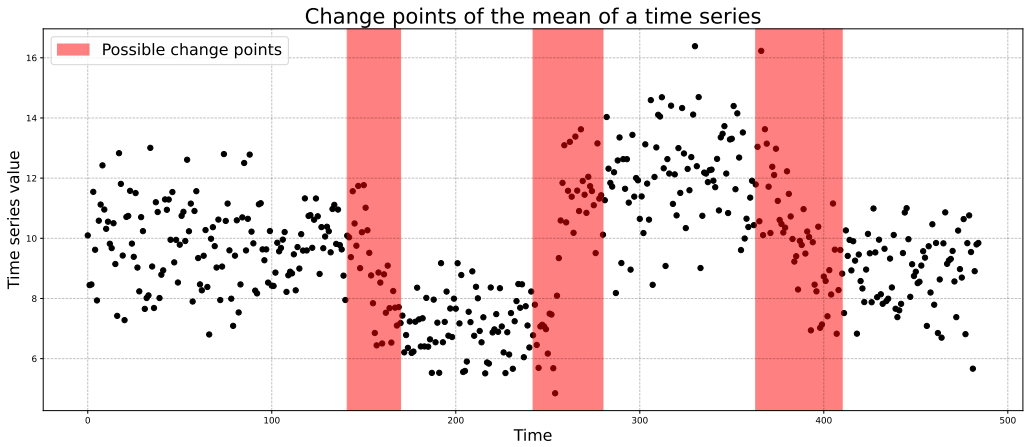
The motivation for the mKPC comes from applications in statistics. In the following, we give a detailed example from change-point detection in time series.

Given a time series $y_1, \ldots, y_n$, 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. Typical change-points for time series occur when the time series changes its expected value (see Figure 2a), its variance (see Figure 2b), or both.

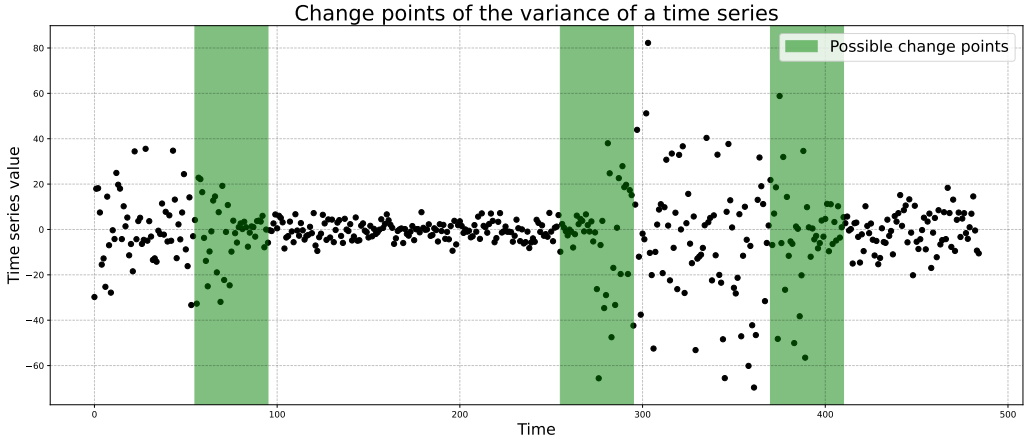
Cappello and Madrid Padilla (2022) introduced a state-of-the-art method, named PRISCA, for detecting changes in the variance of a Gaussian time series. They propose an iterative method which attempts to identify one change point at each iteration. As Figure 2b shows, however, a method identifying one time point for each change point does not give results which are easy to interpret, because there is often considerable uncertainty about when the change takes place. In the figure, this uncertainty is represented by the wide shaded areas. Therefore, at each iteration, PRISCA builds a discrete probability distribution over $\{1, \ldots, n\}$ associating each time point with the probability that it is a change point. An example distribution relative to the first change point is depicted in Figure 3. The height of the bars in the bottom chart corresponds to the probability associated with each time point.

Next, it identifies a level-q *credible set*, i.e., a subset of $\{1, \ldots, n\}$ 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.

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



(a) The expected value of this time series changes three times. Shaded areas show time periods in which, qualitatively, the change appears to be happening.



(b) In this case, the variance of the time series changes while the expected value stays constant.

Figure 2: Example time series which change their expected value and variance. Black points indicate the time series values y_t. Shaded areas represent time periods where, qualitatively, an analyst would expect a change point.

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 2.2, using a unitary scaling factor decidedly simplifies the problem. In the rest of this explanation we will consider, for simplicity, this unit-cost case.

The most straightforward method to build the credible set is perhaps to follow a greedy approach which inserts points by decreasing value of probability until the desired threshold q is met. This criterion was used, for example, by Wang et al. (2020, Supplementary Data, Section A.3). Using this approach, however, can result in a situation in which time points belonging to different change points end up in the same credible set. Figure 4 exemplifies this concept. The points highlighted in yellow in the bottom chart are included in the same credible set, but they are not all associated with the first change point.

To overcome this problem, one must then consider the compactness of the credible set: because each set should identify a single change point, its elements should be "compact" and, ideally, distributed tightly around the real (unknown) change point. This objective can be achieved via compactness constraints. Indeed, once the value of parameter Δ 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. Figure 5 shows how including compactness leads to a better credible set construction.

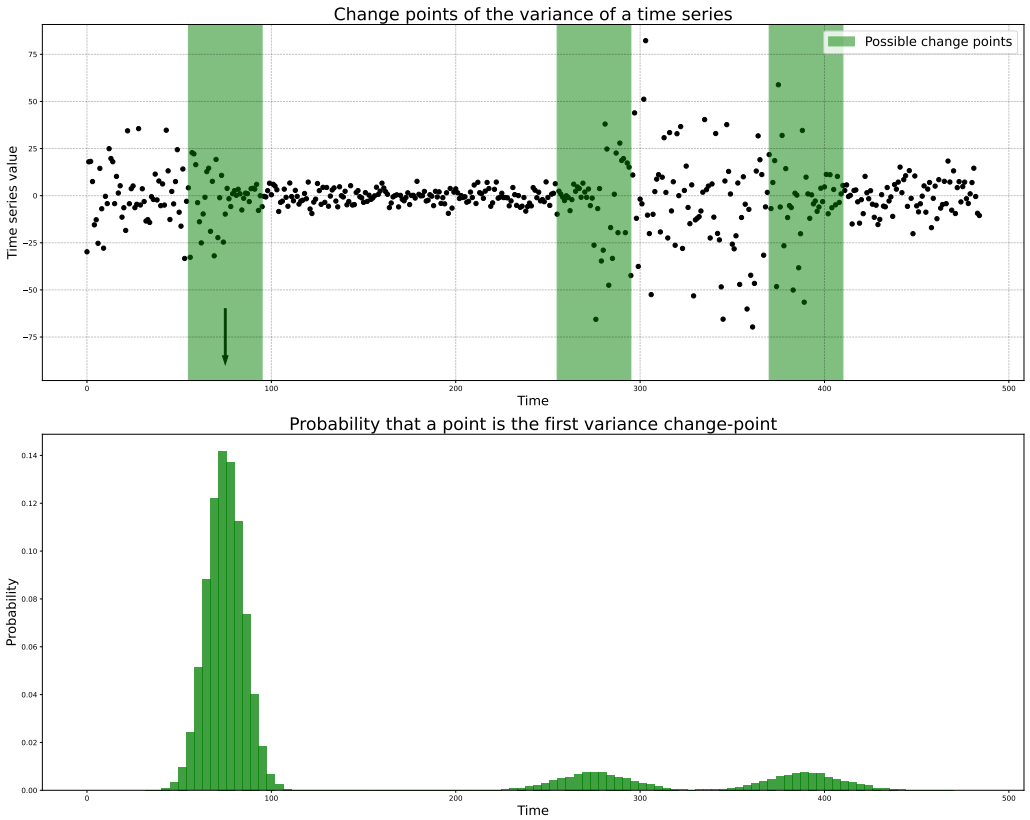


Figure 3: Probabilities associated with each time point and representing how likely the point is to be the first change point of the time series.

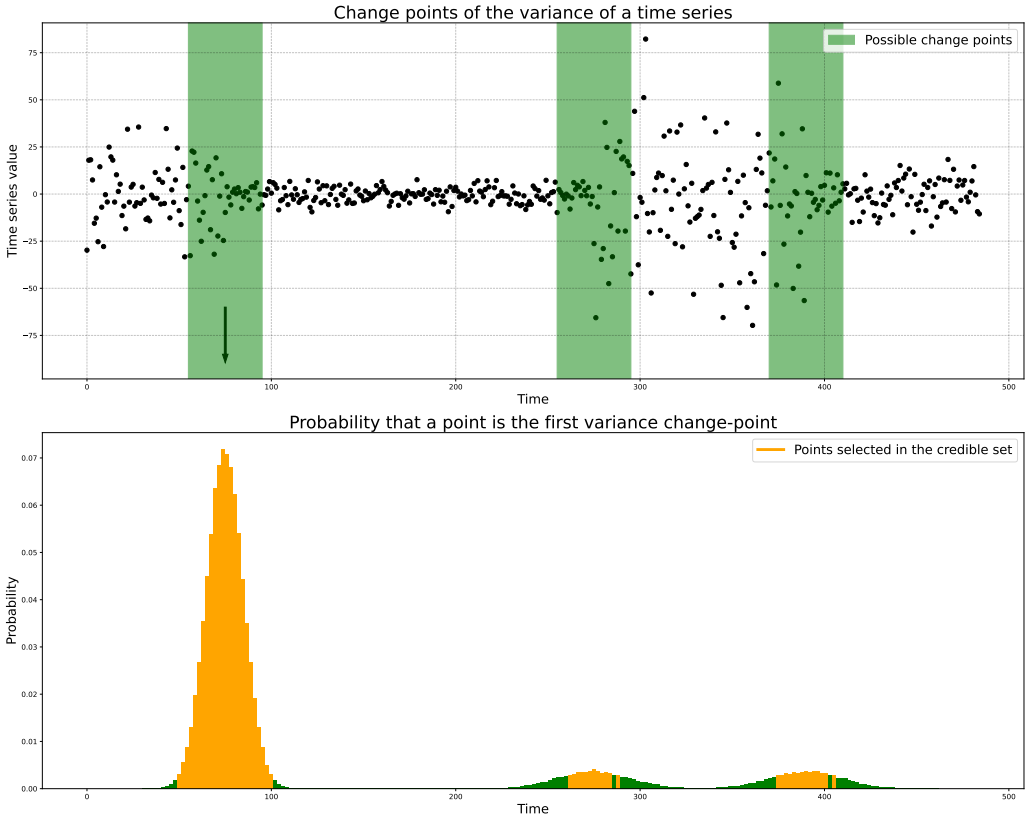


Figure 4: The bottom chart shows a credible set relative to the first change point of the time series in the top chart, when disregarding compactness. The points in the credible set are highlighted in yellow.

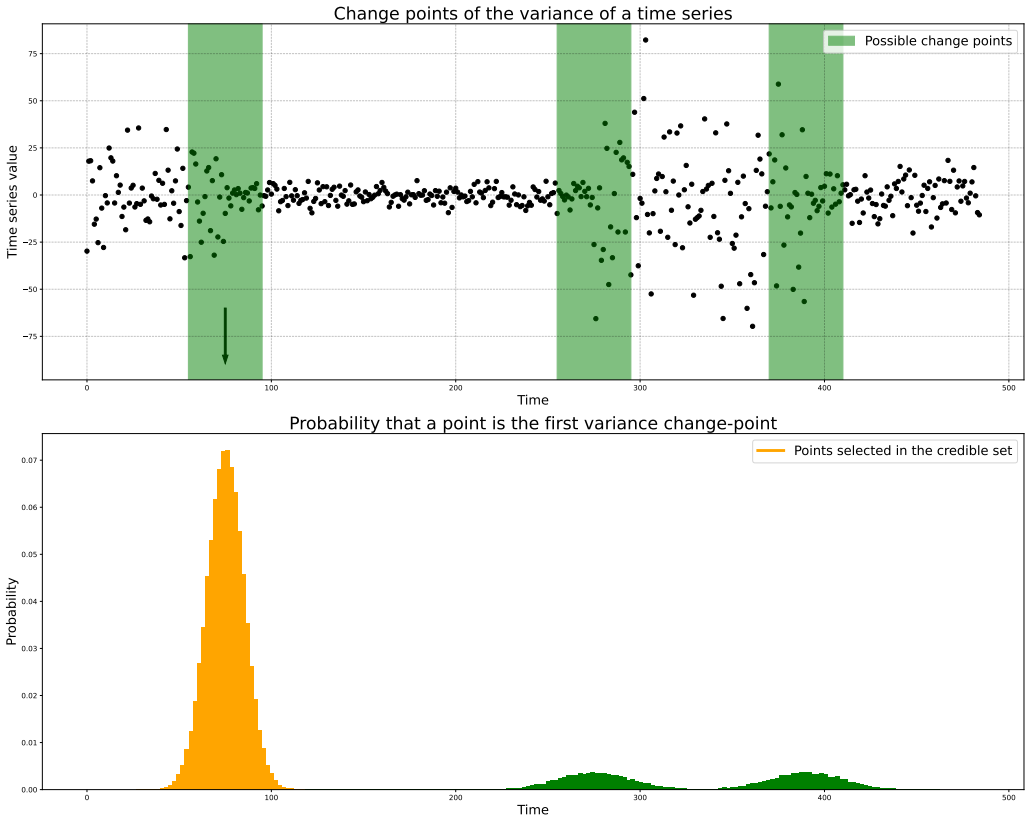


Figure 5: The bottom chart shows a credible set relative to the first change point of the time series in the top chart, considering compactness requirements. The points in the credible set are highlighted in yellow.

2 Formal definition

In this section we give a formal definition of the mKPC by means of an integer programming model and we discuss the complexity of the mKPC and of the unit-cost mKPC (1c-mKPC). As mentioned in Section 1, in fact, the mKPC is $\mathcal{NP}$-complete. In Section 2.2, however, we prove that the 1c-mKPC is solvable in polynomial time.

2.1 Mathematical model

We can formulate the m-KPC as the following integer program, in which binary variable x_j takes value 1 iff the j-th item is selected:

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

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

$$x_i + x_j - 1 \leq \sum_{k=i+1}^{j-1} x_k \qquad \forall i, j \in \{1, \ldots, n\},\ j > i + \Delta \tag{3}$$

$$x_j \in \{0, 1\} \qquad \forall j \in \{1, \ldots, n\}. \tag{4}$$

We denote constraints (3) the *compactness constraints*.

2.2 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 in Section 1.1, however, it can often be the case that all items take unit cost (i.e., $c_j = 1$ for all $i \in \{1, \ldots, n\}$). This problem is denoted as 1c-mKPC and arises, for example, when the user has no prior knowledge of which time instants of a time series are more likely to be change points. 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 $t \in \{1, \ldots, n\}$, 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 $i \in \{1, \ldots, n\}$ and $\ell \in \{1, \ldots, i\}$. Entry $W(i, \ell)$ will contain the maximum weight of a subset of $\{1, \ldots, i\}$ such that the set has size ℓ 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 $i \in \{1, \ldots, n\}$. Furthermore, the following DP recursion is valid:

$$W(i, \ell) = \max_{j \in \left\{[i-\Delta], \ldots, i-1\right\}} \left\{W(j, \ell - 1)\right\} + w_i, \tag{5}$$

where notation $[i - \Delta]$ is used as a shorthand for $\max\{1, i - \Delta\}$. Recursion (5) is valid because of the following observation. Any set of size ℓ having item i as its highest-index element must contain at least one element in $\{[i - \Delta], \ldots, i - 1\}$ 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 $\{1, \ldots, n\}$ of size at most t such that its elements have weight at least c and that satisfies compactness constraints, we must check that

$$\min\left\{\ell \in \{1, \ldots, n\} \mid \exists i \in \{\ell, \ldots, n\} \text{ s.t. } W(i, \ell) \geq q\right\} \leq t. \tag{6}$$

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 (5) we need to compare values in rows $[i - \Delta], \ldots, i - 1$ of column $\ell - 1$, i.e., we perform at most Δ 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 $O(n^3)$. □

3 Related problems

In addition to applications in statistics discussed in Section 1.1, 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 2.2, the mKPC (that is, the Connection Subgraph problem with $T = \emptyset$ and the special structure of graph G) remains $\mathcal{NP}$-complete. The definition of the mKPC as a problem on

a graph gives us an interpretation of inequalities (3) as a special case of the connectivity constraints introduced by Fischetti et al. (2017) to impose connectivity of Steiner trees. However, the special structure of graph G that results when solving the mKPC makes it more efficient to specialize those constraints to the specific problem, without the need to introduce G explicitly. In particular, separation of inequalities (3) is straightforward (see Section 4.2).

As discussed, our compactness constraints can be interpreted as a connectivity requirement on a suited graph. Similar requirements appear in political districting problems, where one has to partition geographic units (e.g., counties or census blocks) to obtain districts for elections. Districts must contain geographically contiguous units and have the same number of inhabitants. Political districting problems are typically defined on a graph where vertices represent the geographic units and have a weight corresponding to the population, and the edges connect units that are contiguous. Hence, the problem consists in partitioning the vertices into subsets having approximately the same weight and inducing connected subgraphs (see, e.g., Ricca, Scozzari and Simeone (2013)).

In a different perspective, Stiglmayr et al. (2022) introduce some measures of robustness for solutions in multi-objective integer linear programming. Here the idea is to select a solution which not only is efficient, but also robust, in the sense that its "closeby" solutions are efficient as well (allowing for a substitution of the selected solution). Closeness of solutions depends on the specific problem, and can be identified as a change of base via a pivot in a linear program, or as a "move" in a combinatorial problem. In any case, close solutions are denoted as adjacent, thus defining a graph. The robustness of each solution is evaluated by analysing its neighbourhood in this graph.

4 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).

4.1 Integer Programming

The first approach consists in solving model (1)–(4) 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 (3) state that if two items lying more than Δ 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 2Δ positions apart, then at least two further items between them shall also be selected. In general, (3) can be strengthened as follows:

$$\left\lfloor \frac{j-i-1}{\Delta} \right\rfloor (x_i + x_j - 1) \leq \sum_{k=i+1}^{j-1} x_k \quad \forall i, j \in \{1, \ldots, n\},\ j > i + \Delta. \tag{7}$$

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 $j \in \{2, \ldots, 1001\}$. 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 Δ 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 $j \in \{2, \ldots, 1001\}$. 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 (3) is satisfied. On the other hand, the strengthened constraint (7) would be violated by such a solution:

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

4.2 On-the-fly constraint generation

Formulation (1)–(4) 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 $x_1^*, \ldots, x_n^*$ violates a compactness constraint. For each item $i \in \{1, \ldots, n\}$ with $x_i^* = 1$, we search the first item $\sigma_i \in \{i + 1, \ldots, n\}$ such that $x_{\sigma_i}^* = 1$. If $\sigma_i > i + \Delta$, then (3) is violated for $j = \sigma_i$ and must be added to the formulation. Otherwise, there is no index j such that constraint (3) 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 $x_1^*, \ldots, x_n^*$, the following procedure determines whether it violates a compactness constraint. For each item $i \in \{1, \ldots, n - \Delta - 1\}$ such that $x_i^* > 0$, let $S = 0$. Then:

1. For each item $k \in \{i + 1, \ldots, i + \Delta\}$, update S with value $S + x_k^*$. If, at some point, $S \geq 1$, then there is no index j for which (3) is violated for the index pair (i, j). We can then move to the next i.
2. Otherwise, start scanning items $j \in \{i + \Delta + 1, \ldots, n\}$ 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 (3) larger or equal than 2 and, thus, the inequality holds. The condition in step 2.a corresponds exactly to a violation of (3), 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 (7) instead of the original (3). In particular, for the fractional case, it is enough to replace the condition in step 2.a with condition

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

4.3 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 σ, a sink node τ, and one node for each item. Overall, $\mathcal{V} = \{\sigma, 1, \ldots, n, \tau\}$. The arc set $\mathcal{A}$ contains:

- Arcs from σ to each node $i \in \{1, \ldots, n\}$.

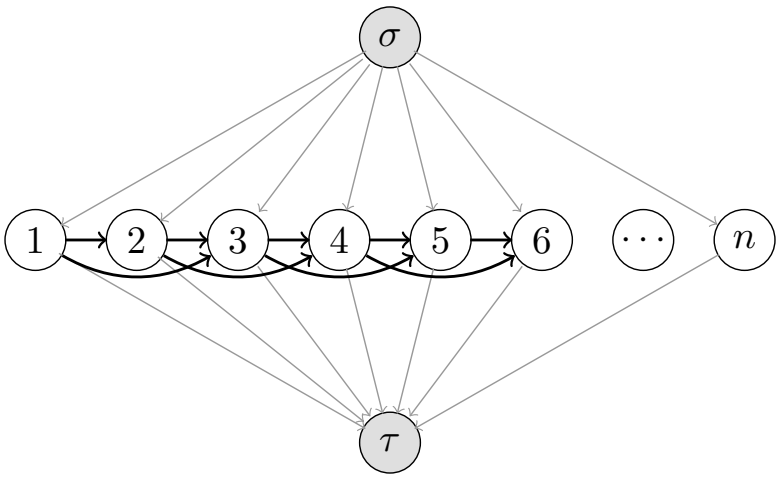


Figure 6: Graph $\mathcal{G}$ used by the labelling algorithm. The graph in the figure depicts an instance with $\Delta = 2$.

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

Figure 6 depicts graph $\mathcal{G}$ when $\Delta = 2$. Thinner arrows represents arcs from σ and to τ, while the thicker ones represents arcs between nodes $\{1, \ldots, n\}$. A feasible solution of the mKPC corresponds to a path in $\mathcal{G}$ starting at σ, ending at τ, 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 σ. 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 $\mathcal{L}' = (i', C', W')$ has components:

$$i' = j, \quad C' = \begin{cases} C + c_j & \text{if } j \in \{1, \ldots, n\} \\ C & \text{if } j = \tau \end{cases}, \quad W' = \begin{cases} W + w_j & \text{if } j \in \{1, \ldots, n\} \\ W & \text{if } j = \tau \end{cases}. \tag{8}$$

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 τ. Analogously, if $W < q$, then it does not make sense to extend that label to τ, 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 τ 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.

4.4 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 $P \subseteq \{1, \ldots, n\}$ 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:

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

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

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

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

5 Computational results

In this section, we report the results of computational experiments to test the effectiveness of the algorithms presented in Section 4. 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 (7).
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 4.4.

5.1 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:

- The Noise method first assigns each item j a weight

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

  where $\mathcal{N}(\lambda, \sigma)$ denotes a normal distribution with location λ and scale σ. 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:

  $$w_j = \frac{w'_j}{\sum_{i=1}^{n} w'_i} \quad \forall j \in \{1, \dots, n\}. \tag{9}$$

  Figure 7 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 $q = 0.95$.

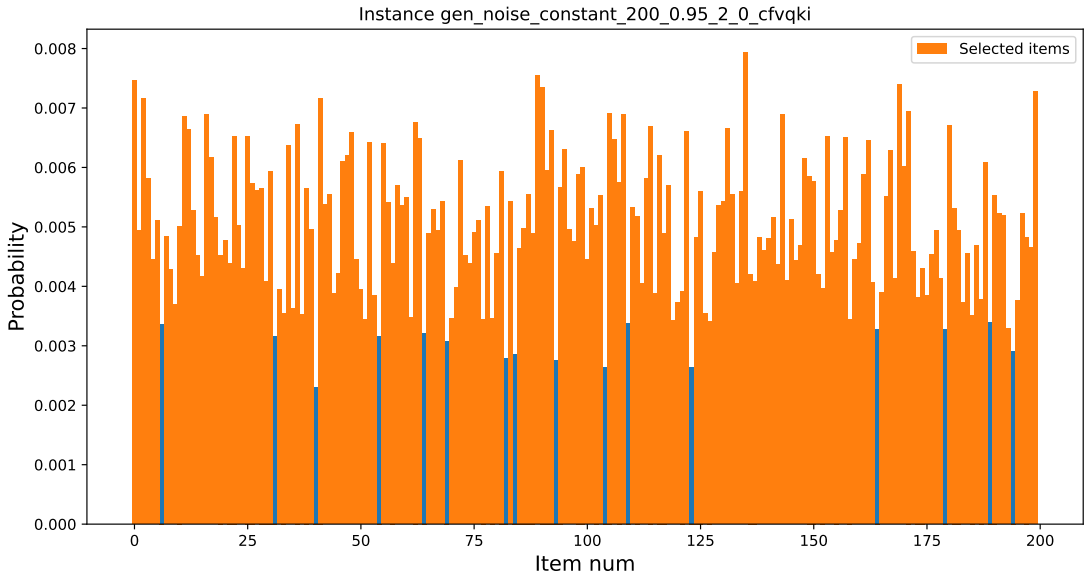


Figure 7: Example NOISE instance with its optimal solution.

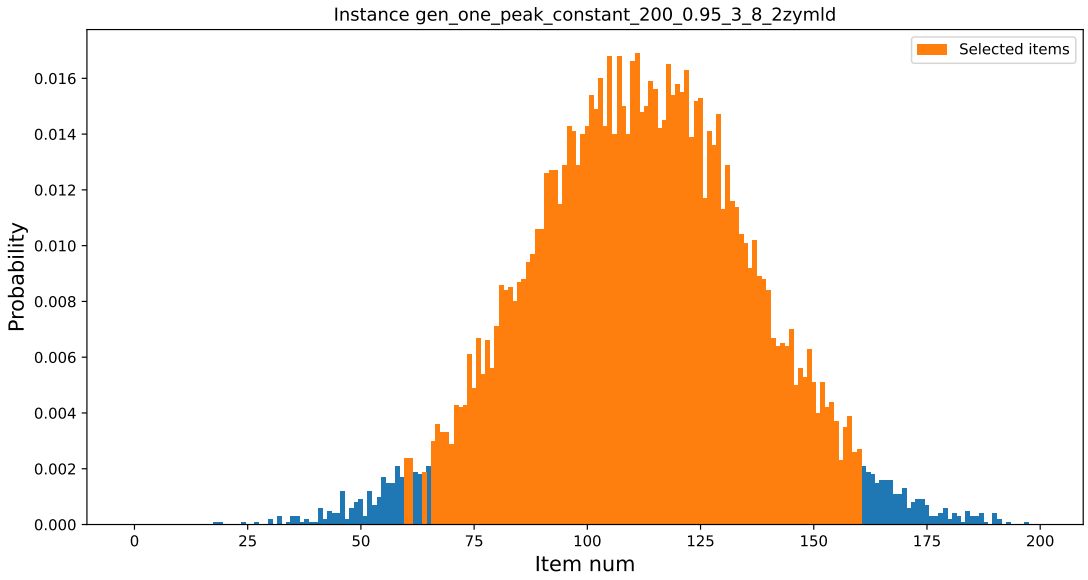


Figure 8: Example ONEPEAK instance with its optimal solution.

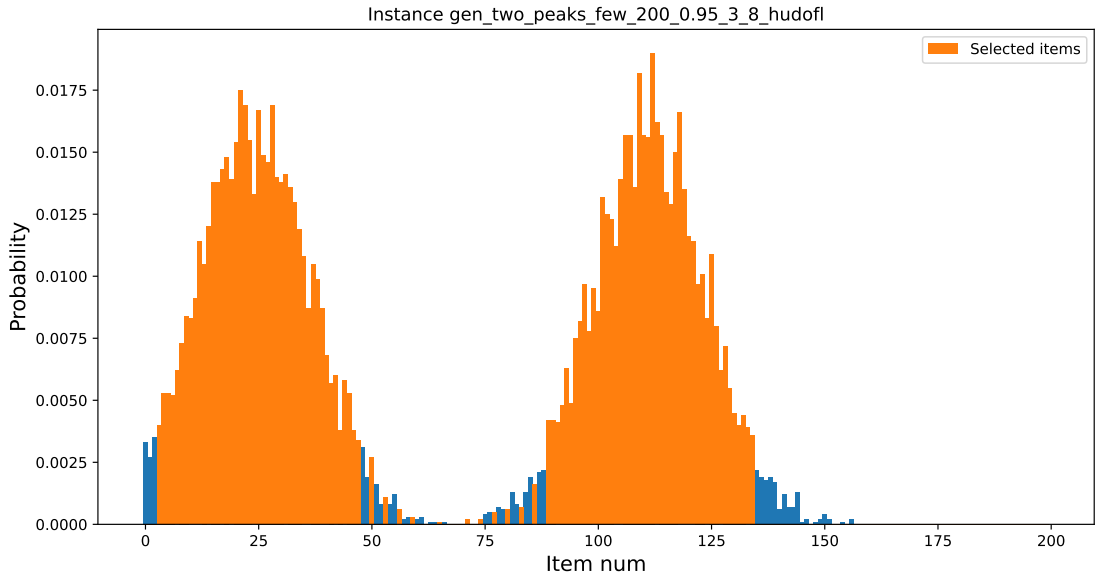


Figure 9: Example TwoPeaks instance with its optimal solution.

- The OnePeak method proceeds as follows. It first chooses a random location λ 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 λ, i.e., an instance similar to the one depicted in Figure 3. To this end, it considers another truncated normal distribution between 1 and n, with location λ 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. (9). Figure 8 shows an example of a OnePeak instance.
- The TwoPeaks methods is similar to OnePeak, except that the histogram is built by sampling from the sum of two truncated normal distributions with locations λ_1 and λ_2, and common scale $\frac{n}{2k}$. Intuitively, λ_1 and λ_2 are the locations of two peaks. 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 9 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 Δ, and three for the cost generation method. Their combination gives 27 parameter combinations using weight generation method Noise. Because we generate 3 instances for each combination, we build 81 Noise instances. Furthermore, for each of these 27 combinations, we have 3 possible values for parameter k, yielding 81 parameter combinations for each of the OnePeak and TwoPeaks weight generation methods. Again, generating 3 instances for each combination, we obtain 243 instances for each of the two methods. Overall, we then construct $81 + 2 \times 243 = 567$ instances.

5.2 Computational experiments

In this section, we present the results of computational experiments on the instances described in Section 5.1. We first investigate the role of strengthened inequalities (7) 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 4.3. 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 (7):

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.

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 (7) the average gaps are roughly reduced by two thirds. We also observe that, on these harder instances, the B&C algorithm loses its advantage

n	Weights	Costs	Compact MIP with (3)	Compact MIP with (7)	B&C with (3)	B&C with (7)
			Time (s)	Time (s)	Time (s)	Time (s)
200	Noise	Constant	3.71	4.82	0.00	0.00
		Few	4.12	6.72	0.01	0.01
		Random	3.92	7.39	0.01	0.01
	OnePeak	Constant	4.74	5.02	0.00	0.01
		Few	4.32	5.25	0.01	0.01
		Random	5.28	6.22	0.05	0.02
400	Noise	Constant	50.60	49.34	0.01	0.01
		Few	56.08	99.85	0.03	0.02
		Random	52.35	63.58	0.03	0.03
	OnePeak	Constant	87.03	104.23	0.01	0.01
		Few	72.92	76.48	0.04	0.02
		Random	72.34	110.54	0.23	0.10
600	Noise	Constant	185.42	187.73	0.01	0.01
		Few	233.83	239.03	0.05	0.02
		Random	243.66	220.93	0.03	0.03
	OnePeak	Constant	466.23	635.56	0.02	0.01
		Few	445.60	455.45	0.33	0.13
		Random	388.22	554.11	0.41	0.19
Overall			152.05	187.17	0.10	0.04

Table 1: Impact of strengthened inequalities (7) 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 with (3)		Compact MIP with (7)		B&C with (3)		B&C with (7)	
		Gap%	Time (s)	Gap%	Time (s)	Gap%	Time (s)	Gap%	Time (s)
200	Constant	5.99	710.39	0.09	492.53	13.53	1746.66	0.83	884.48
	Few	7.63	828.35	0.05	462.61	12.66	2056.36	0.63	627.83
	Random	4.18	578.62	0.54	561.51	8.71	1362.68	0.48	591.01
400	Constant	6.11	1993.72	2.52	1999.99	15.98	2558.45	1.56	1818.52
	Few	10.92	1531.91	2.48	1737.72	13.42	1747.64	3.00	1603.26
	Random	3.51	1079.89	3.41	1915.56	13.21	2606.64	2.69	1350.17
600	Constant	18.88	2131.24	6.46	2147.31	15.97	2133.58	14.07	1740.09
	Few	19.31	2603.57	6.72	2890.47	19.49	2851.15	7.70	2419.43
	Random	12.92	2376.42	6.23	2677.12	15.92	2666.89	4.79	2085.05
Overall		9.94	1537.12	3.17	1653.87	14.13	2008.89	3.81	1428.38

Table 2: Impact of strengthened inequalities (7) 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.

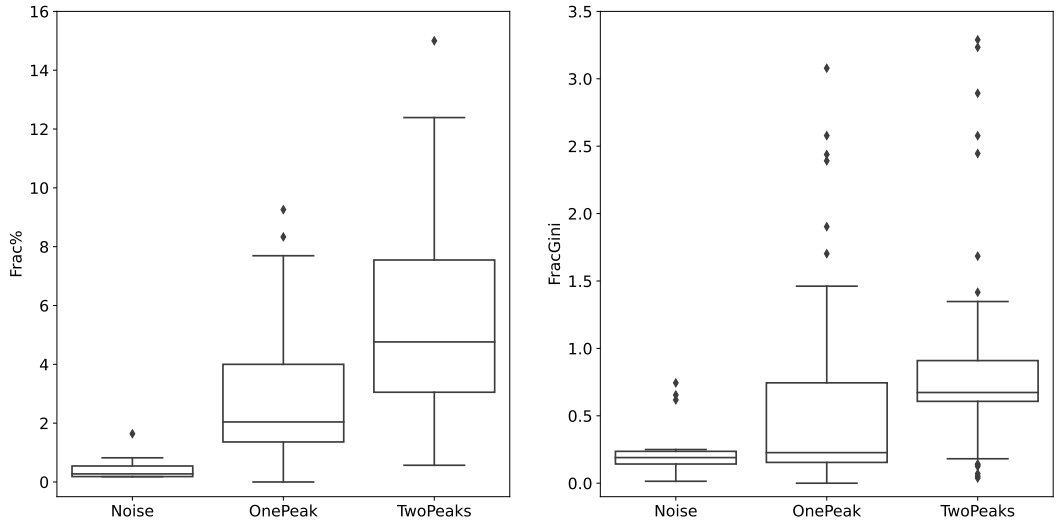


Figure 10: Left: percentage of items selected fractionally in the optimal solution of the continuous relaxation of the mKPC. Right: normalised Gini coefficient showing how close the fractional values are to 0.5 (the higher the value, the closer to 0.5).

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 10 shows boxplots of two metrics relative to the optimal solution of the continuous relaxation of the mKPC, divided by weight generation method. Let $x_1^*, \ldots, x_n^*$ be such a solution. Metric Frac% gives the percentage of items selected fractionally in the solution, i.e.,

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

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

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

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 (7) result in lower gaps, we enable them for both the branch-and-cut algorithm and the compact formulation.

Table 3 reports the following metrics:

n	Weights	Costs	B&C with (7)			Compact MIP with (7)			Labelling			
			Opt%	PGap%	Time (s)	Opt%	PGap%	Time (s)	Feas%	Opt%	PGap%	Time (s)
200	Noise	Constant	100.00	0.00	0.00	100.00	0.00	4.82	100.00	100.00	0.00	1.85
		Few	100.00	0.00	0.01	100.00	0.00	6.72	100.00	100.00	0.00	3.05
		Random	100.00	0.00	0.01	100.00	0.00	7.39	88.89	44.44	26.36	2917.73
	OnePeak	Constant	100.00	0.00	0.01	100.00	0.00	5.02	100.00	100.00	0.00	1.32
		Few	100.00	0.00	0.01	100.00	0.00	5.25	100.00	100.00	0.00	1.59
		Random	100.00	0.00	0.02	100.00	0.00	6.22	100.00	88.89	0.05	125.94
	TwoPeaks	Constant	77.78	0.49	884.48	96.30	0.05	492.53	100.00	96.30	0.13	1.81
		Few	85.19	0.45	627.83	92.59	0.04	462.61	100.00	96.30	0.06	1.59
		Random	88.89	0.01	591.01	88.89	0.00	561.51	100.00	85.19	0.04	144.46
400	Noise	Constant	100.00	0.00	0.01	100.00	0.00	49.34	100.00	100.00	0.00	48.02
		Few	100.00	0.00	0.02	100.00	0.00	99.85	100.00	100.00	0.00	160.17
		Random	100.00	0.01	0.03	100.00	0.01	63.58	66.67	0.00	100.00	3600.00
	OnePeak	Constant	100.00	0.00	0.01	100.00	0.00	104.23	100.00	88.89	0.15	20.09
		Few	100.00	0.00	0.02	100.00	0.00	76.48	100.00	100.00	0.00	34.99
		Random	100.00	0.00	0.10	100.00	0.00	110.54	100.00	74.07	0.52	1975.23
	TwoPeaks	Constant	59.26	1.38	1818.52	55.56	1.42	1999.99	100.00	92.59	0.83	29.94
		Few	55.56	2.12	1603.26	55.56	1.55	1737.72	100.00	92.59	0.79	45.55
		Random	66.67	1.29	1350.17	55.56	1.31	1915.56	96.30	66.67	12.36	2071.43
600	Noise	Constant	100.00	0.00	0.01	100.00	0.00	187.73	100.00	100.00	0.00	234.71
		Few	100.00	0.00	0.02	100.00	0.00	239.03	100.00	100.00	0.00	1090.45
		Random	100.00	0.01	0.03	100.00	0.01	220.93	66.67	0.00	100.00	3600.00
	OnePeak	Constant	100.00	0.00	0.01	100.00	0.00	635.56	100.00	96.30	0.02	207.80
		Few	100.00	0.00	0.13	100.00	0.00	455.45	100.00	85.19	0.04	282.16
		Random	100.00	0.00	0.19	100.00	0.00	554.11	88.89	40.74	25.05	3337.04
	TwoPeaks	Constant	51.85	14.12	1740.09	44.44	5.46	2147.31	100.00	100.00	3.32	231.36
		Few	33.33	13.26	2419.43	29.63	4.83	2890.47	100.00	88.89	2.89	345.75
		Random	44.44	4.20	2085.05	29.63	4.62	2677.12	81.48	18.52	39.37	3485.78
Overall			83.95	1.73	620.83	83.25	0.92	815.75	97.18	83.42	6.24	697.20

Table 3: Comparison of the MIP-based approaches (the branch-and-cut and the compact formulation) with the labelling algorithm presented in Section 4.3, on the S2 instances.

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.
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 τ, 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 τ, then we do not even have a feasible solution to compute PGap%. Therefore, we introduce the additional column 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 PGap% of 100 % and a Time (s) of 3600 s 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 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 4.4. 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.

6 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

Algorithm	n	Opt%	PGap%	Time (s)
B&C with (7)	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 (7)	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 4.3)	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 4.4)	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.

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.

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, to Lorenzo Cappello for fruitful discussions on the problem, and to three anonymous reviewers for their constructive comments. 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.

Ricca, F., A. Scozzari and B. Simeone (2013). 'Political districting: from classical models to recent approaches'. In: *Annals of Operations Research* 204, pp. 271–299. DOI: 10.1007/s10479-012-1267-2.

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.

Stiglmayr, M., J. R. Figueira, K. Klamroth, L. Paquete and B. Schulze (2022). 'Decision space robustness for multi-objective integer linear programming'. In: *Annals of Operations Research* 319, pp. 1769–1791. DOI: 10.1007/s10479-021-04462-w.

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.