Optimization Letters
https://doi.org/10.1007/s11590-025-02252-y

ORIGINAL PAPER

Algorithms and complexity results for the 0–1 knapsack problem with group fairness

Enrico Malaguti[1] · **Paolo Paronuzzi**[1] · **Alberto Santini**[2,3]

Received: 3 March 2025 / Accepted: 10 September 2025

Abstract

This paper presents an exact algorithm for a generalisation of the classical 0–1 Knapsack Problem, called the Knapsack Problem with Group Fairness. In this problem, items are partitioned into classes, and fairness constraints affect the number of items that can or must be chosen from each class. The problem was introduced by Patel et al. (in: Dignum (eds) Proceedings of the 20th International Conference on Autonomous Agents and MultiAgent Systems. International Foundation for Autonomous Agents and Multiagent Systems, 2021), where approximation algorithms are discussed. This paper describes the first exact solution approaches for the Knapsack Problem with Group Fairness, based on integer linear programming formulations and a dynamic programming algorithm. The latter allows us to establish the complexity of the problem. Finally, a set of computational experiments on benchmark instances derived from the knapsack literature compare the effectiveness of the alternative solution approaches.

Keywords Packing · Knapsack problems · Fairness · Integer linear programming · Dynamic programming

✉ Alberto Santini
alberto.santini@upf.edu

Enrico Malaguti
enrico.malaguti@unibo.it

Paolo Paronuzzi
paolo.paronuzzi@unibo.it

[1] Department of Electrical, Electronic and Information Engineering, Alma Mater Studiorum University of Bologna, Viale del Risorgimento, 2, 40136 Bologna, Italy

[2] Department of Economics and Business, Universitat Pompeu Fabra, Carrer Trias-Fargas, 25–27, 08005 Barcelona, Spain

[3] Data Science Centre, Barcelona School of Economics, Carrer Trias-Fargas, 25–27, 08005 Barcelona, Spain

Published online: 06 October 2025

1 Introduction

This paper presents an exact algorithm for a generalisation of the classical 0–1 Knapsack Problem (KP) in which items are partitioned into classes, and fairness constraints affect the number of items that can or must be chosen from each class. In the KP [30], one is given n items, each one having a profit $p_j \in \mathbb{N}$ and weight $w_j \in \mathbb{N}$, to be packed into a knapsack of capacity $c \in \mathbb{N}$ while maximising the packed profit. Patel et al. [33] introduced a family of problems, collectively called KP with Group Fairness (KPGF), in which the items are partitioned into ℓ classes $J_1, \ldots, J_\ell$. The main idea behind the KPGF is that the items packed in the knapsack should represent each class fairly. To this end, the authors consider three different problems in which each class has associated upper and lower bounds on the total weight, total profit or number of items chosen from the class. In this paper, we generalise this idea further by associating a resource consumption value $h_j \in \mathbb{N}$ to each item and imposing lower and upper bounds on the total resource consumption of packed items of each class. The three problems introduced by Patel et al. [33] then correspond to the special cases in which, for each item j, $h_j = w_j$, $h_j = p_j$, and $h_j = 1$, respectively.

The KPGF can be used to determine a fair allocation of economic resources. Pael et al. [33] and the references therein list some of its most important applications. Here, we highlight one use case which is particularly important due to its timeliness and social impact: participatory budgeting (PB). A participatory budget is a decision-making process in which ordinary citizens decide how to allocate part of a local government's budget. Contemporary PB was pioneered in the '90 s by the Brazilian city of Porto Alegre [18, 32] and is now used in thousand of major and small cities and communities around the world, such as New York City [41], Barcelona [3], Bologna, Stuttgart and Zaragoza [28]. In a typical PB process at the municipal level, citizens propose interventions they would like to see implemented in their neighbourhood, such as a new bike lane or renovating a public space. Municipality technicians then filter the technically feasible interventions and assign an estimated budget to each one. Then, a public voting phase starts, during which citizens can vote on one or multiple projects they would like implemented. At the end of this phase, the municipality selects which interventions to perform, limited by the available budget and considering citizens' preferences. The problem of selecting the subset of interventions to implement can be modelled as a KP in which the items are the interventions, the capacity is the available budget, the weights are the intervention costs, and the profits are the number of votes gathered. Still, such a system could result in one or a few neighbourhoods receiving most interventions while others do not. Therefore, the municipality is usually interested in establishing rules that ensure that the implemented projects affect all parts of the city. In this case, the problem becomes a KPGF in which the item classes are the neighbourhoods, and many different fairness principles can be used. For example, fairness could be quantified by the number of projects (resulting in a KPGF with all $h_j = 1$), the budget (resulting in a KPGF with $h_j = w_j$), or some other criterion, such as the number of positively affected citizens (resulting in a KPGF in which the values h_j do not coincide with other problem data).

In this paper, we generalise the KPGF introduced by Patel et al. [33] and present models and an exact solution algorithm for this problem. Moreover, we settle the open

question on the complexity of the KPGF by proving that it is weakly $\mathcal{NP}$-hard. After reviewing the related literature in Sect. 2, we introduce a compact and an extended formulation based on Dantzig–Wolfe reformulation in Sect. 3. The extended formulation has a number of variables which is exponential in n. However, in Proposition 3.1 we show that only a pseudopolynomial number of them are required to solve the KPGF to optimality. We exploit this property to devise a solution algorithm outperforming the direct application of a state-of-the-art mixed-integer programming solver to the compact formulation, as shown by the results presented in Sect. 4. In Sect. 6 we summarise the main findings and propose further research directions. Finally, we publish the first instance set for the KPGF and make the source code of our solver available [38].

2 Literature review

The KPGF was introduced by Patel et al. [33]. The authors consider six problems stemming from the combination of two base problems and three notions of fairness. The two base problems are the classical KP and the min-Knapsack problem [2, 17]. As mentioned in the introduction, the three notions of fairness impose bounds on the number of items chosen in each class, their total weight, or their total profit. They prove that finding a feasible KPGF solution is weakly $\mathcal{NP}$-hard when the resource consumption coincides with the weight or the profit. They also provide one approximation algorithm for each of the six problems considered. Each algorithm can produce a sub-optimal solution, violate a resource consumption bound, violate the knapsack capacity, or present a combination of these three properties. The authors do not implement the proposed algorithms, which are mostly of theoretical interest, and no computational experiments are carried out. Compared with [33], our paper presents the first exact solution approaches for the KPGF. Furthermore, our approaches solve a generalisation of the problem with an arbitrary resource and establishes complexity results for this generalised version.

Our work continues a recent research stream on generalisations of the KP; see, e.g., Malaguti et al. [27]; Clautiaux et al. [15]; Schäfer et al. [39]; Al-douri et al. [1]; Monaci et al. [31]; Boeckmann et al. [11]; Santini and Malaguti [37]; see also the recent surveys by Cacchiani et al. [12, 13].

In the rest of this section, we place our work in the current literature along three axes: knapsack variants with similar features (Sect. 2.1), other optimisation problems with fairness (Sect. 2.2), and other optimisation problems with similar applications (Sect. 2.3).

2.1 Knaspack variants with similar features

Cappanera and Trubian [14] introduced the Multidemand Multidimensional Knapsack Problem (MMKP). In this problem, profits can be either positive or negative, and each item is associated with multiple resources with both upper and lower bounds. The KPGF can be seen as a special case of the MMKP. In particular, we can transform any KPGF instance with ℓ classes into an MMKP instance with $\ell + 1$

resources in the following way. The KPGF objective function is valid for the MMKP. The KPGF's capacity constraint corresponds to an MMKP resource constraint in which the resource is the weight, the lower bound is zero, and the upper bound is the capacity. Finally, we create an MMKP resource for each KPGF class. Given a class k, the resource consumption of an item j in the MMKP is h_j if $j \in J_k$ in the KPGF or zero otherwise. The resource upper and lower bounds are $\underline{h}_k$ and $\bar{h}_k$, respectively.

Cappanera and Trubian [14] propose a heuristic two-stage Tabu Search algorithm for the MMKP. In the first stage, the algorithm explores infeasible solutions until a feasible one is found. In the second stage, the feasible solution is improved upon. The authors also remark that general-purpose mixed-integer solvers struggle to find feasible solutions for instances with a few hundred of items and tens of inequalities. We are not aware of any specialised exact algorithm for the MMKP.

Xu [45] studies the Knapsack Problem with a Minimum Filling Constraint (KPMFC). The KPMFC extends the classical KP by imposing a minimum weight to pack. It can be seen as a special case of the MMKP when there is only one resource, the weight, with both a lower and upper bound. The author proves that when the lower bound equals the upper bound, the problem of finding a feasible solution for the KPMFC is weakly $\mathcal{NP}$-hard; the proof is by reduction from the partition problem [21]. Xu [45] then considers the special case where the bound is strictly smaller than the capacity and develops an approximation algorithm. The paper, however, does not include computational results; therefore, an empirical comparison with the method of Cappanera and Trubian [14] is impossible. The KPMFC can also appear in algorithms that solve other optimisation problems. For example, the Dynamic Programming (DP) algorithm presented by Furini et al. [20] for the Minimum-Cost Maximal Knapsack Problem requires solving multiple instances of the KPMFC.

Bettinelli et al. [9] consider a related problem in the bin packing context, where one has to minimise the total cost of the bins required to pack all items. Bins can be of different types; the type chosen determines the cost, capacity, and minimum weight that must be packed into each used bin. The authors propose a branch-and-price algorithm whose pricing subproblem is a restricted version of the MMKP with a single resource and both lower and upper bounds. The authors perform heuristic pricing and then resort to a black-box mixed-integer problem when the heuristic fails to produce a negative reduced cost column. The authors also mention that they compared the black-box solver with a DP algorithm, ultimately favouring the former, but they do not provide any details about the DP algorithm.

Finally, we mention the Multidimensional Multiple-Choice Knapsack Problem. In this problem, items are partitioned into classes, and each item is associated with a profit and several resources. The objective is to select one item in each class to maximise the collected profit and ensure that the lower and upper bounds on each resource are respected. This problem was introduced, with a different name, by van de Velde and Worm [42]. The current state-of-the-art exact algorithm is the YACA procedure by Mansini and Zanotti [29].

2.2 Optimisation problems with fairness constraints

Fairness is a timely topic that has been subject to investigation in many areas of knowledge, including combinatorial optimisation problems. It usually arises in resource allocation, where a third party must assign desirable resources to a set of agents. In general, there are several ways of defining and quantifying fairness (see, e.g., [26]). For example, one can minimise the difference between the agents who receive the most and the least resources. In assignment problems, a popular criterion is envy-freeness [19, 44], i.e., devising an assignment such that each agent believes their assignment is no worse than any other agent's. In the KPGF, agents are classes, resources are items, and assigning a resource to an agent means packing an item of the corresponding class in the knapsack. The notion of fairness in the KPGF states that no agent can receive too many or too few resource units.

We remark here that the term "Group Fairness" in the KPGF as introduced by Patel et al. [33] slightly differs from the common usage in the literature. In its most common meaning [6, 16], group fairness refers to the concept of fair allocation among groups of agents rather than among individual agents. The specific way in which group fairness is modelled can vary greatly. For example, this notion can apply *on top* of individual-agent fairness or *instead of* it; in particular applications [7], group fairness can also apply to overlapping groups of agents (that is, an agent can belong to multiple groups). On the other hand, the KPGF does not include fairness for groups of agents, i.e., for groups of classes taken together.

Finally, we mention that another stream of research investigates the "price of fairness", i.e., the loss in resource allocation efficiency when including fairness constraints. This concept was introduced by Bertsimas et al. [8] and has been widely employed since then (see, e.g., [4, 24, 25]).

2.3 Optimisation problems with similar applications

Resource allocation problems often arise in the public sector. For example, the government of Singapore has faced the problem of assigning public housing to families while ensuring that each neighbourhood houses a balanced mix of ethnic groups [5].

Relevant to our key application in PB, Serramia et al. [40] study the problem of optimally selecting projects in Barcelona's participatory budget. Similar to our problem, the authors use a knapsack-like model in which the proposals are the items, the costs are the weights, and the votes received are the profits. The authors do not consider fairness; they focus on avoiding the simultaneous selection of conflicting projects. For example, if a project proposes to create a children's playground in a public square and another proposes to build a skate park in the same space, only one can be implemented. The resulting optimisation problem is a 0–1 Knapsack Problem with Conflict Graph (KPCG; see, e.g., [10]). The authors solve this problem with a black-box solver and compare the results with the current selection method used in Barcelona. This method ranks the projects in descending order of votes and selects them up to the available budget, skipping a project if another incompatible one has already been selected. In optimisation terms, this corresponds to a greedy KPCG heuristic sorting the items by profit. The authors show that an approach based on optimal

KPCG solutions can increase the number of accepted proposals and their total support, measured as the total number of votes gathered by implemented projects.

3 Mathematical formulations

A compact integer programming formulation for the KPGF uses binary variables $x_j \in \{0,1\}$ for each $j \in \{1,\dots,n\}$ taking value one if and only if item j is packed. The formulation reads as follows.

$$\max \quad \sum_{j=1}^{n} p_j x_j \tag{1a}$$

$$\text{subject to} \quad \sum_{j=1}^{n} w_j x_j \leq c \tag{1b}$$

$$\sum_{j \in J_k} h_j x_j \geq \underline{h}_k \qquad \forall k \in \{1,\dots,\ell\} \tag{1c}$$

$$\sum_{j \in J_k} h_j x_j \leq \bar{h}_k \qquad \forall k \in \{1,\dots,\ell\} \tag{1d}$$

$$x_j \in \{0,1\} \qquad \forall j \in \{1,\dots,n\}. \tag{1e}$$

The objective function maximises the collected profit, while constraint (1b) ensures that the knapsack capacity is respected. Constraints (1c) and (1d) ensure that each class's lower and upper bound on the resource consumption are satisfied.

An alternative formulation for the problem can be obtained by applying a Dantzig–Wolfe reformulation (see, e.g., [43]) to model (1a)–(1e). The resulting model optimises over the convex hull of vectors satisfying constraints (1b)–(1d) which are encoded, for each class k, by the collection $\mathcal{R}_k$ of valid packings of items of class k. A packing in $\mathcal{R}_k$ is a subset of items $R \subseteq J_k$ such that

$$\underline{h}_k \leq \sum_{j \in R} h_j \leq \bar{h}_k \quad \text{and} \quad \sum_{j \in R} w_j \leq c. \tag{2}$$

Remark that, if $\underline{h}_k = 0$, then $\emptyset \in \mathcal{R}_k$, i.e., it is possible that no item of class k is packed. Let p_R, w_R and h_R be the profit, weight and resource consumption associated with a generic packing R, i.e.,

$$p_R = \sum_{j \in R} p_j, \qquad w_R = \sum_{j \in R} w_j, \quad \text{and} \quad h_R = \sum_{j \in R} h_j.$$

For any class k and packing $R \in \mathcal{R}_k$, let $\eta_R \in \{0,1\}$ be a variable taking the value one if the items in R are the ones packed for class k. Then, the reformulation for the KPGF reads Dantzig–Wolfe

$$\max \quad \sum_{k=1}^{\ell} \sum_{R \in \mathcal{R}_k} p_R \eta_R \tag{3a}$$

$$\text{subject to} \quad \sum_{k=1}^{\ell} \sum_{R \in \mathcal{R}_k} w_R \eta_R \leq c \tag{3b}$$

$$\sum_{R \in \mathcal{R}_k} \eta_R = 1 \qquad \forall k \in \{1, \ldots, \ell\} \tag{3c}$$

$$\eta_R \in \{0,1\} \qquad \forall k \in \{1, \ldots, \ell\},\ \forall R \in \mathcal{R}_k. \tag{3d}$$

The objective function maximises the packed profit, constraint (3b) ensures that the knapsack's capacity is not exceeded, and constraint (3c) imposes that exactly one packing is chosen for each class.

This formulation defines a Multiple-Choice Knapsack Problem (MCKP), a knapsack problem where items are partitioned into classes, and exactly one item per class must be selected. The problem is known to be weakly $\mathcal{NP}$-hard. In particular, an MCKP with $\bar{n}$ items and capacity c can be solved in $O(\bar{n}c)$ operations by using a DP algorithm (see [23, 30]).

For each class k, each packing $R \in \mathcal{R}_k$ of the KPGF corresponds to a MCKP item. Because the number of packings is exponential in n, (3a)–(3d) defines a MCKP with a number of items (say, $\bar{n}$) exponential in the number of KPGF items (n). However, in the following proposition, we show that, given the KPGF's special structure, considering a subset of variables of pseudo-polynomial size is sufficient to find an optimal solution. In the rest of this section, we show that this subset can be generated using a DP algorithm with pseudo-polynomial time complexity.

Proposition 3.1 *For each class k, consider the sub-collection $\bar{\mathcal{R}}_k \subset \mathcal{R}_k$ including at most one packing for each integer value $W \in \{1, \ldots, c\}$, namely, a packing R maximising p_R and having $\underline{h}_k \leq h_R \leq \bar{h}_k$ and $w_R = W$, if one exists. The optimal solution of formulation (3a)–(3d) restricted to the variables associated with these sub-collections is optimal for the KPGF.*

Proof For each class k, the sub-collection $\bar{\mathcal{R}}_k$ contains one optimal packing for each amount of the capacity which can be allocated to the class. □

For each class k, the sub-collection $\bar{\mathcal{R}}_k$ can be enumerated by the DP algorithm presented in the next section.

3.1 Dynamic programming algorithm for packing enumeration

Let $n_k = |J_k|$ be the number of items of class k, and $J_k = \{j_{k1}, \ldots, j_{kn_k}\}$. Let $\zeta_k(H, W, m)$ be the largest profit that can be achieved with the first m items in J_k (i.e., $\{j_{k1}, \ldots, j_{km}\}$) such that the selected items have total weight no larger than W and total resource consumption equal to H, and let $\rho_k(H, W, m) \subseteq J_k$ be an associated packing. Furthermore, denote with $H_k(W)$ the resource consumption of a packing whose total weight is W, $m = n_k$, and the profit is maximum for the given weight, i.e.,

$$H_k(W) = \underset{H \in \{\underline{h}_k, \ldots, \bar{h}_k\}}{\arg\max} \ \zeta_k(H, W, n_k).$$

The sub-collection $\bar{\mathcal{R}}_k$ is defined as:

$$\bar{\mathcal{R}}_k = \left\{ \rho_k\big(H_k(W), W, n_k\big) : \ W \in \{1, \ldots, c\} \right\}. \tag{4}$$

The values of $\zeta_k(H, W, m)$ and the associated packings $\rho_k(H, W, m)$ can be computed via DP by using the following recursion:

$$\zeta_k(H, W, m) = \max \begin{cases} \zeta_k(H, W, m-1), \\ p_{j_{km}} + \zeta_k(H - h_{j_{km}}, W - w_{j_{km}}, m-1). \end{cases} \tag{5}$$

In (5), we build $\bar{\mathcal{R}}_k$ by choosing, for each feasible weight W, the packing giving the maximum profit and respecting the resource consumption bounds (if any).

The first case corresponds to not packing item j_{km}. In contrast, j_{km} is packed in the second case, yielding its profit and decreasing the available capacity and the target resource consumption level. Remarking that, in the recursion, at least one and possibly all three arguments of ζ_k decrease, one can provide the following base cases that are valid for small enough values of H, W, and m:

$$\zeta_k(H, W, m) = -\infty \quad \text{if } H < \min_{i=1,\ldots,m} h_{j_{ki}} \text{ or } H > \sum_{i=1}^{m} h_{j_{ki}}, \tag{6}$$

$$\zeta_k(H, W, m) = -\infty \quad \text{if } W < \min_{i=1,\ldots,m} w_{j_{ki}}, \tag{7}$$

$$\zeta_k(H, W, 1) = \begin{cases} p_1 & \text{if } H = h_{j_{k1}} \text{ and } w_{j_{k1}} \le W \\ -\infty & \text{otherwise.} \end{cases} \tag{8}$$

Condition (6) refers to two trivial cases in which it is not possible to meet the exact resource level H with items $\{j_{k1}, \ldots, j_{km}\}$ either because H is too small or too large compared with the resources associated with the items. We remark that it might be impossible to meet level H in other cases not included in (6), namely when no combination of items can yield exactly H resource units. Still, condition

$H < \min_{i=1,\ldots,m} h_{j_{ki}}$ is sufficient to obtain a valid base case for the DP algorithm because it becomes active when the value of H is small enough. Condition (7) refers to the case when it is impossible to respect the capacity bound because W is too small. Condition (8) considers the first item j_{k1} only and checks whether taking it as the only item yields a feasible solution. Finally, we use the above DP algorithm to settle the question of the KPGF's complexity.

Proposition 3.2 *The KPGF is weakly NP-hard.*

Proof Let $h = \max_k \{\bar{h}_k - \underline{h}_k\}$. For each class k, we can compute $\bar{\mathcal{R}}_k$ in $O(n_k ch)$, i.e., $O(nch)$ for all classes.

The input for the MCKP has size $\left|\bigcup_{k=1}^{\ell} \bar{\mathcal{R}}_k\right| = O(\ell c)$. Therefore, the DP algorithm for the MCKP needs $O(\ell c^2)$ operations.

The overall complexity of our approach is, thus, $O(nch) + O(\ell c^2)$. □

3.2 Class-specific capacities

For a given class k', let $\hat{w}_{k'}$ be the minimum weight of a subset of items of this class consuming an amount of resource between $\underline{h}_{k'}$ and $\bar{h}_{k'}$. The maximum capacity that the items of a class k can occupy in the knapsack of the KPGF is therefore reduced by the values of $\hat{w}_{k'}$ of all other classes. By computing the values $\hat{w}_{k'}$ for all classes, we define per-class capacities $\hat{c}_k : -c - \sum_{k' \in \{1,\ldots,l\} \setminus k} \hat{w}_{k'}$ which can be used in (2) and (5), replacing capacity c.

The minimum weight of subsets of items of a class k can be found by solving the following integer program, in which variable $y_j \in \{0, 1\}$ (for $j \in J_k$) indicates that item j is part of the set.

$$\min \quad \sum_{j \in J_k} w_j y_j \tag{9a}$$

$$\text{subject to} \quad \sum_{j \in J_k} h_j y_j \geq \underline{h}_k \tag{9b}$$

$$\sum_{j \in J_k} h_j y_j \leq \bar{h}_k \tag{9c}$$

$$y_j \in \{0, 1\} \qquad \forall j \in J_k. \tag{9d}$$

Model (9a)–(9d) describes a min-Knapsack problem, with the addition of the classical KP constraint (9c). If problem (9a)–(9d) is infeasible for some class or its optimal solution value is larger than c, then the entire KPGF is infeasible. Indeed, in this case it is impossible to select items from such a class while respecting the resource

consumption bounds and the knapsack's capacity. Moreover, the KPGF is infeasible if $\hat{c}_k < 0$ for any class k because, in this case, the knapsack does not have enough capacity to accommodate items of all classes. We remark that any lower bound on $\hat{w}_{k'}$, for a given class k', can be used to set the value of $\hat{c}_k$ for $k \neq k'$. Using a lower bound in the computation of $\hat{w}_{k'}$ would result in a larger (and, therefore, looser) value for $\hat{c}_k$. In our case, a black-box solver optimally solves problems (9a)–(9d) almost instantaneously. Therefore, our implementation uses values $\hat{c}_k$ instead of c.

4 Computational results

In this section, we report on computational experiments comparing our approach based on the extended formulation with the direct application of a state-of-the-art solver to the compact formulation. Regarding the extended formulation, we first generate the pseudopolynomial number of variables required to solve the model to optimality by using the DP algorithm presented in Sect. 3.1. Recall that the resulting problem is an MCKP. We solve this problem by means of the algorithm described in Pisinger [35] and available at [34]. We solve the compact formulation by using Gurobi version 11.0.0. Both approaches are tested on a machine with four Intel i7 processors running at 2.60 GHz and 16 GB RAM, running in single-thread mode. All algorithms have a 1 h time limit.

In the remainder of this section, we describe how we generated the test instances and provide a computational comparison between the two approaches described above. The instances, their generator, the solver's code, and the scripts used to generate the figures of this section are available at [38].

4.1 Instance generation

We generate our KPGF instances from base KP instances. We use the thirteen generators introduced by Pisinger [36] to generate the KP instances. The generators are: *almost strongly correlated, circle, inverse strongly correlated, MSTR, profit ceiling, span uncorrelated, span weakly correlated, span strongly correlated, strongly correlated, subset sum, uncorrelated, uncorrelated similar, and weakly correlated*. All the generators except *uncorrelated similar* require an additional parameter (denoted *R*) corresponding to the order of magnitude of the values used for the profits and the weights. Following Pisinger [36], we use 1000 and 10,000 for this value, while the *uncorrelated similar* instances use a hard-coded value of 100,000.

Our generation procedure starts from a KP instance with n items, profits p_j, weights w_j, and capacity c and transforms it into a KPGF instance with ℓ classes ($\ell \leq n$). We first assign to each item i the class $1 + (i|\ell)$, where $i|\ell$ denotes the remainder of the integer division of i by ℓ. This way, we ensure that all classes are non-empty and their sizes are roughly equal. Second, we generate the resource bounds $\underline{h}_k$ and $\bar{h}_k$ by drawing them independently from the uniform integer distributions $\mathcal{U}(50, 150)$ and $\mathcal{U}(150, 250)$, respectively. Third, we generate the resource consumption values h_j. Denote with $r = \left(\sum_{j=1}^{n} w_j\right)/c$ the ratio between the total item weights and

the capacity. The resource consumption of an item j belonging to class k is set to $h_j = \min\{\lceil ru/|J_k| \rceil, \bar{h}_k\}$, where notation $\lceil \cdot \rceil$ denotes integer rounding and u is drawn from the uniform integer distribution $\mathcal{U}(125, 175)$. If the total resource consumption of items in a class is strictly smaller than the class's resource lower bound, we discard the instance because it is trivially infeasible.

The above procedure aims to maintain tight capacity and resource bounds, making the instances more challenging while avoiding generating too many infeasible instances. First, remark that r^{-1} is a rough approximation of the fraction of items that can be packed in the knapsack due to the capacity constraint. For example, if the total weight of the items is twice as large as the capacity, we can expect that only half of the items can be packed. Thus, we expect $r^{-1}n$ items to be packed in total and that $r^{-1}|J_k|$ items of class k will be packed. Remark that $\mathbb{E}[\underline{h}_k] = 100$ and $\mathbb{E}[\bar{h}_k] = 200$. Using the procedure described above, the expected resource consumption of a subset of size $r^{-1}|J_k|$ of class-k items is 150, thus increasing the likelihood that a solution respecting both the capacity and each class's resource bounds can be found.

There are $12 \times 2 + 1 = 25$ combinations of generator and parameter R: two per each generator using this parameter, plus one for the *uncorrelated similar* generator, which does not use it. For each of the 25 combinations, we consider values of $n \in \{50, 100, 200, 500, 1000, 2000\}$ and $\ell \in \{20, 100, 500\}$ and consider the twelve (n, ℓ) combinations with $n > \ell$. Finally, we generate ten instances for each of the $25 \times 12 = 300$ combinations of generator, R, n and ℓ, for a total of 3000 instances. Out of these, 1787 are feasible, and the remaining 1213 are provably infeasible. The results presented in the rest of this section refer to the feasible instances.

4.2 Computational results

Figure 1 shows a performance profile comparing the extended and the compact formulations. The x axis reports the time required for solving an instance, normalised by the time required by the fastest of the two algorithms. The y axis reports the percentage of instances solved to optimality within the given runtime. The dark blue line refers to the compact formulation, and the light orange line refers to the extended formulation. Interestingly, the compact formulation outperforms the extended one when instances are easy. Indeed, an analysis of the detailed results shows that the compact

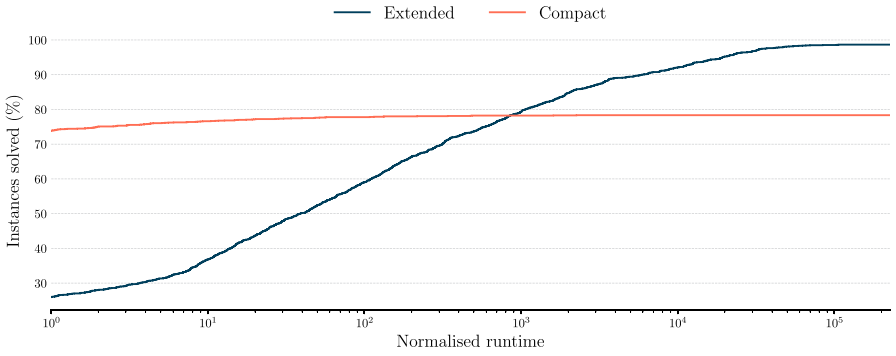

Fig. 1 Performance profile comparing the compact and extended formulations

formulation solves approximately 69% of the instances within one second. In contrast, the extended formulation only solves about 14% of the instances in the same time. However, this 69% of the instances constitutes the majority of the instances the compact formulation can solve, no matter how much time it is given (up to a maximum of 1 h). Indeed, the compact formulation can only solve 1400 out of the 1787 instances, i.e., 78.34%. By contrast, the extended formulation solves more instances when given more time: in total, it solves 1763 out of the 1787 instances, i.e., 98.66%.

To summarise, the compact formulation solves fewer instances, but when it does, it takes less time. By contrast, the extended formulation solves more instances and tends to use a larger fraction of the available time. Figures 2 and 3 provide more details and characterise the hard instances for the compact and the extended formulation, respectively. Each figure contains three charts; each chart displays the percentage of instances solved to optimality when aggregating them according to three criteria. The first criterion (left chart) is the KP generator used to build the base knapsack instance. The second criterion (top right chart) is the number of items n, and the third one (bottom right chart) is the data range R, which determines the order of magnitude of weights, profits and capacity. Figure 2 also shows the average percentage gaps reported by Gurobi when timing out while solving the compact formulation. The gaps are computed over the open instances and are displayed above each bar for which there are open instances. It is not possible to compute the gaps for the extended formulation because the timeouts always occur during the column generation phase (Sect. 3.1), whereas, once this phase is completed, the MCKP solver only takes a few seconds to produce the optimal solution. When the timeout occurs while generating

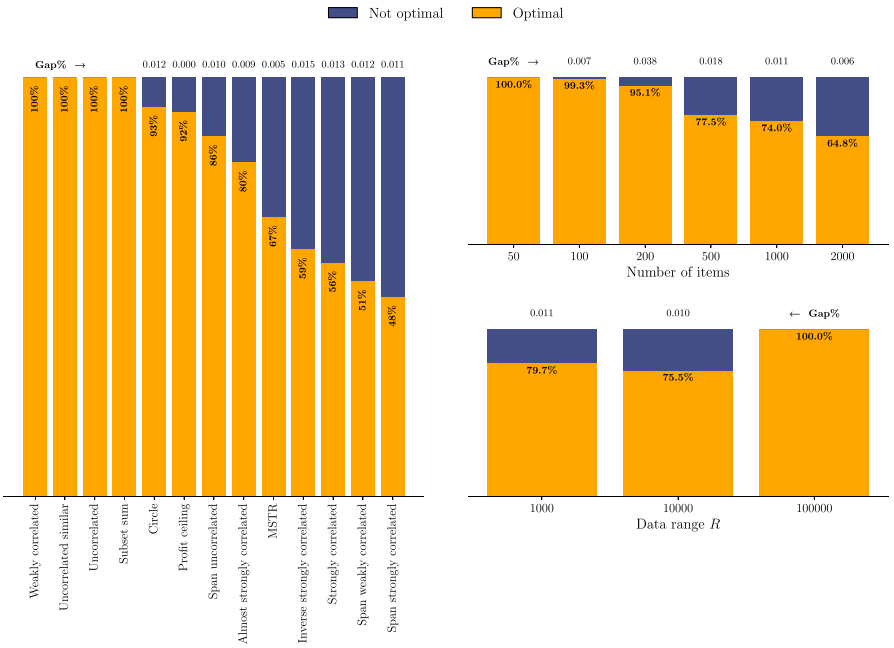

Fig. 2 Results summary for the compact formulation

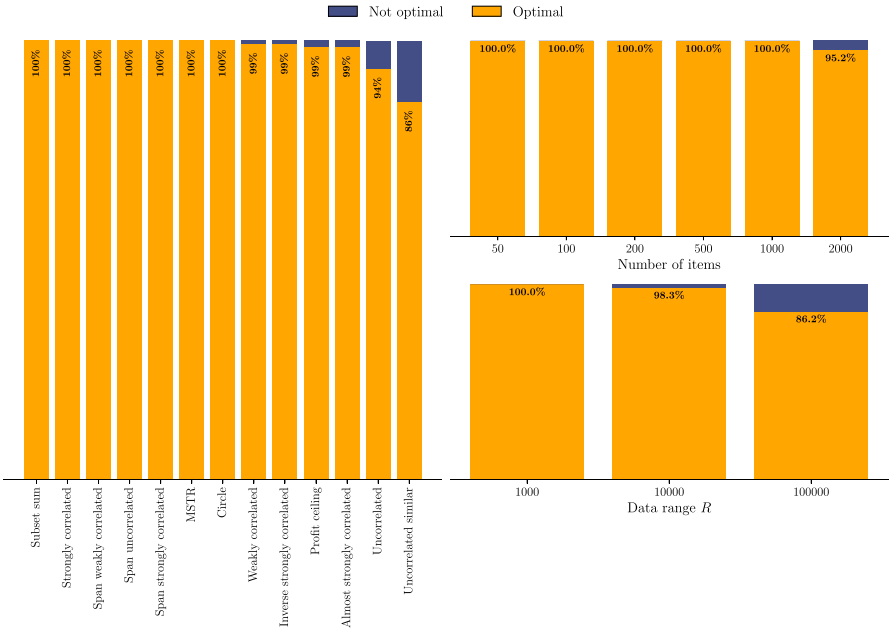

Fig. 3 Results summary for the extended formulation

packings, we do not have a valid MCKP instance to solve, and therefore, no valid bounds. Indeed, if we removed the 1 h time limit and let the packing generation procedure complete, we would be able to solve all instances in at most 6972 s using the extended formulation.

Figure 2 shows that the compact formulation is sensitive to the base knapsack instance type, solving all instances of type *weakly correlated*, *uncorrelated similar*, *uncorrelated* and *subset sum*, but less than two thirds of the *inverse strongly correlated*, *strongly correlated*, and *span weakly correlated* ones, and less than half of the *span strongly correlated* instances. As expected, increasing the number of items also increases the instance difficulty: the compact formulation solves all instances with 50 items but less than two-thirds of those with 2000 items. Finally, increasing the data range parameter from 1000 to 10,000 makes the instances harder. The compact formulation solves all instances with a data range of 100,000 to optimality. However, recall that following Pisinger [36], only instances of type *uncorrelated similar* use this data range, and these instances are particularly easy to solve with the compact formulation despite their larger data range.

The extended formulation generally outperforms the compact one, solving more instances. However, we notice some complementarity among the two approaches: the *uncorrelated similar* instances, which were among the easiest to solve with the compact formulation, are the hardest for the extended one. The reason is that, as remarked in Proposition 3.2, the algorithm we use to solve the extended formulation has complexity $O(nch) + O(\ell c^2)$. The *uncorrelated similar* instances have a capacity c one or two orders of magnitude larger than the other instances (100,000 vs. 1000 or 10,000), thus affecting the algorithm's runtime.

At the same time, we remark that the compact formulation's gaps on the open instances are generally small, due to the large profits appearing in the [36] instances. When classifying the instances by generator, the *inverse strongly correlated* instances result in the largest average gaps (0.015%). Interestingly, using a capacity of 1 000 or 10 000 does not impact the average gaps significantly. Regarding the number of items, while larger instances are harder to solve for the compact formulation, they do not result in larger average gaps. As mentioned above, this is due to the fact that instances with more items have larger profits, resulting in larger absolute values for the upper and lower bounds.

With respect to the extended formulation, as mentioned above, if we increased the time limit from 1 to 2 h, this formulation would close all instances. This further confirms the previous observation prompted by the analysis of the performance profile (Fig. 1): the extended formulation seems to benefit from longer computation times more than the compact formulation.

We conclude this section by reporting that new hard knapsack instances have been recently proposed by Jooken et al. [22]. These instances are inherently difficult and have even larger values for the item profits and the knapsack capacities compared to the instances of Pisinger [36]. In particular, they have capacities of one million, 100 million, and 10 billion. We generated a set of KPGF instances starting from the [22] instances with a procedure analogous to that described in Sect. 4.1. We used the 01–KP instances with the capacity of the order of magnitude relevant for our application, i.e., one million (see also Sect. 5). Interestingly, the extended formulation could solve all 1685 feasible instances in at most 376 s, whereas the compact one only solved 81.5% of the instances within the 1 h time limit. The generator, the generated instances, the results, and their analysis are available at [38].

5 Case study

To illustrate the practical relevance of the KPGF, we analyse the participatory budgeting allocation process that took place in Barcelona, Spain, in 2020 and whose projects were implemented during 2020–2023. This case study shows that considering fairness in knapsack problems can result in alternative and improved solutions supporting real-world decision-making. The city of Barcelona allocates specific budgets to each district and implements a selection process in which projects are approved based on their vote counts until the district's budget is exhausted.

We examine how alternative optimisation approaches may lead to improved outcomes while respecting the same total budget. Specifically, we compare the solution implemented by Barcelona's municipality (As-is) with two alternative methods. The first method (KP) optimally solves each district's separate knapsack problem, with the district's budget serving as the knapsack capacity. The second method (KPGF) aims to achieve a fair allocation across districts using the KPGF. In the KPGF approach, the total budget, i.e., the sum of the districts' budgets, is used as the capacity of the knapsack, while we removed the resource upper bound constraints. The resource lower bound for each class is set equal to the budget allocated to the corresponding district in the As-is implementation. This ensures every district receives at least as

Table 1 Solution comparison for a real participatory budgeting process (Barcelona 2020–2023)

District	Budget	As–is			KP			KPGF		
		# Proj	Votes	Cost	# Proj	Votes	Cost	# Proj	Votes	Cost
Ciutat Vella	3400	9	12,656	3080	11	14,254	3380	11	13,498	3085
Eixample	3000	8	18,674	2995	12	22,512	2995	13	23,683	3245
Gràcia	2400	8	14,576	2225	8	14,576	2225	8	14,576	2225
Horta Guinardó	3200	7	9069	3187	10	11,133	3157	10	11,006	3197
Les Corts	2000	8	6870	1970	11	8241	1948	11	8205	2023
Nou Barris	3600	9	11,043	3573	13	13,826	3460	13	14,272	3654
Sant Andreu	3000	9	11,142	2973	12	13,762	2853	13	14,308	3113
Sant Martí	3600	7	13,625	3480	11	18,434	3459	12	18,940	3669
Sants Montjuïc	3600	5	12,960	3570	11	21,441	3514	12	22,144	3639
Sarrià	2200	6	4309	2093	8	4992	2183	8	4860	2143
Total	30,000	76	114,924	29,145	107	143,171	29,173	111	145,492	29,992

much funding as in the currently implemented solution, while allowing optimisation to redistribute additional resources. Other policy choices could determine lower bounds differently.

Table 1 compares the three solution approaches. Column "#Proj" reports the number of financed projects, "Votes" indicates the number of votes received by selected projects, and "Cost" reports the total cost of the chosen projects in thousands of euros. Each row displays the results for a single district, while the last row contains aggregate sums across all districts. We did not report computing times, as they are negligible and out of the scope of this analysis.

Our analysis reveals that using an optimisation-based approach leads to substantial improvements in overall satisfaction (measured as the number of votes received by the selected projects). Specifically, the KPGF solution demonstrates superior performance across all metrics, achieving the largest total votes (145 492) and selecting more projects (111) than both the As-is (76 projects, 114 924 votes) and KP (107 projects, 143 171 votes) approaches. The larger total cost (29 992 thousands of euros) also indicates efficient budget utilization.

6 Conclusions

This work has generalised the Knapsack Problem with Group Fairness (KPGF) introduced by Patel et al. [33]. We proved that the problem is weakly $\mathcal{NP}$-hard and devised two mathematical formulations for it. The first one is a compact formulation using n variables, where n is the number of items in the instance. The second is an extended formulation using a number of variables which is exponential in n. However, we proved that a subset of variables of pseudopolynomial size is sufficient to solve the problem optimally. We provided a constructive algorithm based on DP to build this subset. Computational experiments on instances with 50–2000 items demonstrated the superiority of our approach based on the DP algorithm and the extended formulation.

One of the main applications of the KPGF is in participatory budgeting. In Sect. 5, we present a case study showing how approaches based on the KPGF can improve

the currently implemented solutions. Future research could incorporate additional relevant constraints for this application. For example, Serramia et al. [40] suggest that a decision-support tool for participatory budgeting should include conflict constraints that prevent mutually incompatible projects from being selected simultaneously.

Funding Alberto Santini was funded by the Spanish Ministry of Science, Innovation and Universities and by the European Social Fund Plus (ESF+) through grant RYC2022-035269-I via MCIN/AEI/10.13039/501100011033 of the "Ramón y Cajal" programme and through grant PID2024-161521OA-I00 of the programme "Proyectos de Generación de Conocimiento 2024" of the Spanish Ministry of Science, Innovation and Universities. Enrico Malaguti and Paolo Paronuzzi were funded by the Air Force Office of Scientific Research under award number FA8655-25-1-7013.

Data availability The instances used in this paper, their generator, the solver's code, and the scripts used to generate the figures of this section are available at [38].

References

1. Al-douri, T., Hifi, M., Zissimpoulos, V.: An iterative algorithm for the max–min knapsack problem with multiple scenarios. Oper. Res. Int. J. **21**, 1355–1392 (2021). https://doi.org/10.1007/s12351-019-00463-7
2. Babat, L.: Linear functionals on the n-dimensional unit cube. Rep. Acad. Sci. Soviet Union **221**, 761–762 (1975)
3. Barandiaran, X.E., Calleja-López, A., Monterde, A., Romero, C.: A technopolitical network for participatory democracy: the future of a collective platform. In: Decidim, A Technopolitical Network for Participatory Democracy: Philosophy, Practice and Autonomy of a Collective Platform in the Age of Digital Intelligence, pp. 119–133. Springer Nature Switzerland, Cham (2024)
4. Bei, X., Lu, X., Manurangsi, P., et al.: The price of fairness for indivisible goods. Theory Comput. Syst. **65**, 1069–1093 (2021). https://doi.org/10.1007/s00224-021-10039-8
5. Benabbou, N., Chakraborty, M., Ho, X.V., et al.: Diversity constraints in public housing allocation. In: Dastani M, Sukthankar G, André E, et al (eds) Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems. International Foundation for Autonomous Agents and Multiagent Systems, 973–981 (2018)
6. Benabbou, N., Chakraborty, M., Elkind, E., et al.: Fairness towards groups of agents in the allocation of indivisible items. In: Kraus S (ed) Proceedings of the 28th International Joint Conference on Artificial Intelligence. AAAI Press, 95–101 (2019). https://doi.org/10.24963/ijcai.2019/14
7. Bera, S., Chakrabarty, D., Flores, N., et al.: Fair algorithms for clustering. In: Wallach H, Larochelle H, Beygelzimer A, et al (eds) Proceedings of the 32nd Conference on Advances in Neural Information Processing Systems (NeurIPS). Curran Associates, 4954–4965 (2019)
8. Bertsimas, D., Farias, V., Trichakis, N.: The price of fairness. Oper. Res. **59**(1), 17–31 (2011). https://doi.org/10.1287/opre.1100.0865
9. Bettinelli, A., Ceselli, A., Righini, G.: A branch-and-price algorithm for the variable size bin packing problem with minimum filling constraint. Ann. Oper. Res. **179**, 221–241 (2008). https://doi.org/10.1007/s10479-008-0452-9
10. Bettinelli, A., Cacchiani, V., Malaguti, E.: A branch-and-bound algorithm for the knapsack problem with conflict graph. INFORMS J. Comput. **29**(3), 457–473 (2017). https://doi.org/10.1287/ijoc.2016.0742
11. Boeckmann, J., Thielen, C., Pferschy, U.: Approximating single- and multi-objective nonlinear sum and product knapsack problems. Dis. Optim. **48**, 100771 (2023). https://doi.org/10.1016/j.disopt.2023.100771
12. Cacchiani, V., Iori, M., Locatelli, A., et al.: Knapsack problems—An overview of recent advances. Part I: Single knapsack problems. Comput. Oper. Res. **143**, 105692 (2022). https://doi.org/10.1016/j.cor.2021.105692

13. Cacchiani, V., Iori, M., Locatelli, A., et al.: Knapsack problems—An overview of recent advances. Part II: Multiple, multidimensional, and quadratic knapsack problems. Comput. Oper. Res. **143**, 105693 (2022). https://doi.org/10.1016/j.cor.2021.105693
14. Cappanera, P., Trubian, M.: A local-search-based heuristic for the demand-constrained multidimensional knapsack problem. INFORMS J. Comput. **17**, 82–98 (2005). https://doi.org/10.1287/ijoc.1030.0050
15. Clautiaux, F., Detienne, B., Guillot, G.: An iterative dynamic programming approach for the temporal knapsack problem. Eur. J. Oper. Res. **293**, 442–456 (2021). https://doi.org/10.1016/j.ejor.2020.12.036
16. Conitzer, V., Freeman, R., Shah, N., et al.: Group fairness for the allocation of indivisible goods. In: Peoceedings of the 33th AAAI Conference on Artificial Intelligence. AAAI Press, 1853–1860 (2019). https://doi.org/10.1609/aaai.v33i01.33011853
17. Csirik, J., Frenk, H., Labbé, M., et al.: Heuristics for the 0–1 min-knapsack problem. Acta Cybernet. **10**(1–2), 15–20 (1991)
18. Sousa, S.B.: Participatory budgeting in Porto Alegre: toward a redistributive democracy. Polit. Soc. **26**(4), 461–510 (1998). https://doi.org/10.1177/0032329298026004003
19. Foley, D.K.: Resource allocation and the public sector. PhD Thesis, Yale University (1966)
20. Furini, F., Ljubić, I., Sinnl, M.: An effective dynamic programming algorithm for the minimum-cost maximal knapsack packing problem. Eur. J. Oper. Res. **262**(2), 438–448 (2017). https://doi.org/10.1016/j.ejor.2017.03.061
21. Garey, M., Johnson, D.: Computers and Intractability. W. H, Freeman and Company (1979)
22. Jooken, J., Leyman, P., Causmaecker, P.: A new class of hard problem instances for the 0–1 knapsack problem. Eur. J. Oper. Res. **301**(3), 841–854 (2022)
23. Kellerer, H., Pferschy, U., Pisinger, D.: The multiple-choice knapsack problem. In: Kellerer H, Pferschy U, Pisinger D (eds) Knapsack Problems. Springer, 317–347 (2004). https://doi.org/10.1007/978-3-540-24777-7_11
24. Lodi, A., Malaguti, E., Stier-Moses, N.E.: Efficient and fair routing for mesh networks. Math. Program. **124**, 285–316 (2010). https://doi.org/10.1007/s10107-010-0356-8
25. Lodi, A., Sankaranarayanan, S., Wang, G.: A framework for fair decision-making over time with time-invariant utilities. Eur. J. Oper. Res. **319**, 456–467 (2023). https://doi.org/10.1016/j.ejor.2023.11.030
26. Luss, H.: On equitable resource allocation problems: a lexicographic minimax approach. Oper. Res. **47**(3), 361–378 (1999). https://doi.org/10.1287/opre.47.3.361
27. Malaguti, E., Monaci, M., Paronuzzi, P., et al.: Integer optimization with penalized fractional values: the knapsack case. Eur. J. Oper. Res. **273**, 874–888 (2019). https://doi.org/10.1016/j.ejor.2018.09.020
28. Manes-Rossi, F., Brusca, I., Levy Orelli, R., et al.: Features and drivers of citizen participation: insights from participatory budgeting in three European cities. Public Manag. Rev. **25**(2), 201–223 (2023). https://doi.org/10.1080/14719037.2021.1963821
29. Mansini, R., Zanotti, R.: A core-based exact algorithm for the multidimensional multiple choice knapsack problem. INFORMS J. Comput. **32**(4), 1061–1079 (2020). https://doi.org/10.1287/ijoc.2019.0909
30. Martello, S., Toth, P.: Knapsack Problems. Wiley (1990)
31. Monaci, M., Pike-Burke, C., Santini, A.: Exact algorithms for the 0–1 time-bomb knapsack problem. Comput. Oper. Res. **145**, 105848 (2022). https://doi.org/10.1016/j.cor.2022.105848
32. Novy, A., Leubolt, B.: Participatory budgeting in Porto Alegre: social innovation and the dialectical relationship of state and civil society. Urban Studi. **42**(11), 2023–2036 (2005). https://doi.org/10.1080/00420980500279828
33. Patel, D., Khan, A., Louis, A.: Group fairness for knapsack problems. In: Dignum F, Lomuscio A, Endriss U, et al (eds) Proceedings of the 20th International Conference on Autonomous Agents and MultiAgent Systems. International Foundation for Autonomous Agents and Multiagent Systems, 1001–1009 (2021). https://doi.org/10.5555/3463952.3464069
34. Pisinger, D.: David Pisinger's optimization codes. (1995a). https://web.archive.org/web/20250219094749/http://hjemmesider.diku.dk/ pisinger/codes.html
35. Pisinger, D.: A minimal algorithm for the multiple-choice knapsack problem. Eur. J. Oper. Res. **83**, 394–410 (1995). https://doi.org/10.1016/0377-2217(95)00015-I
36. Pisinger, D.: Where are the hard knapsack problems? Comput. Oper. Res. **32**, 2271–2284 (2005). https://doi.org/10.1016/j.cor.2004.03.002
37. Santini, A., Malaguti, E.: The min-knapsack problem with compactness constraints and applications in statistics. Eur. J. Oper. Res. **312**, 385–397 (2024). https://doi.org/10.1016/j.ejor.2023.07.020

38. Santini, A., Paronuzzi, P.: Source code and instances for the 0–1 knapsack problem with group fairness. (2025). https://doi.org/10.5281/zenodo.16600726, https://github.com/alberto-santini/kpgf
39. Schäfer, L., Dietz, T., Barbati, M., et al.: The binary knapsack problem with qualitative levels. Eur. J. Oper. Res. **289**, 508–514 (2021). https://doi.org/10.1016/j.ejor.2020.07.040
40. Serramia, M., Lopez-Sanchez, M., Rodríguez-Aguilar, J., et al.: Optimising participatory budget allocation: The decidim use case. In: Sabater-Mir J, Torra V, Aguiló I, et al (eds) Artificial Intelligence Research and Development, Frontiers in Artificial Intelligence and Applications, vol 319. IOS Press, 193–202 (2019). https://doi.org/10.3233/FAIA190124
41. Su, C.: From Porto Alegre to New York city: participatory budgeting and democracy. New Polit. Sci. **39**, 67–75 (2017). https://doi.org/10.1080/07393148.2017.1278854
42. van de Velde, S., Worm, J.: Multi-period planning of road maintenance: a multiple-choice multiple-knapsack problem. Tech. Rep. 94-6, Laboratory of Production and Operations Management, University of Twente (1994)
43. Vanderbeck, F., Wolsey, L.A.: Reformulation and decomposition of integer programs. In: Jünger M, Liebling TM, Naddef D, et al (eds) 50 Years of Integer Programming 1958-2008: From the Early Years to the State-of-the-Art. Springer Berlin Heidelberg, 431–502 (2010). https://doi.org/10.1007/978-3-540-68279-0_13
44. Varian, H.: Equity, envy, and efficiency. J. Econ. Theory **9**, 63–91 (1974). https://doi.org/10.1016/0022-0531(74)90075-1
45. Xu, Z.: The knapsack problem with a minimum filling constraint. Nav. Res. Logist. **60**, 56–63 (2013). https://doi.org/10.1002/nav.21520