Group Fairness for Knapsack Problems

Deval Patel  
Indian Institute of Science, Bengaluru, India  
devalpatel@iisc.ac.in

Arindam Khan  
Indian Institute of Science, Bengaluru, India  
arindamkhan@iisc.ac.in

Anand Louis  
Indian Institute of Science, Bengaluru, India  
anandl@iisc.ac.in

ABSTRACT

We study the knapsack problem with group fairness constraints. The input of the problem consists of a knapsack of bounded capacity and a set of items. Each item belongs to a particular category and has an associated weight and value. The goal of this problem is to select a subset of items such that all categories are fairly represented, the total weight of the selected items does not exceed the capacity of the knapsack, and the total value is maximized. We study the fairness parameters such as the bounds on the total value of items from each category, the total weight of items from each category, and the total number of items from each category. We give approximation algorithms for these problems. We also give experimental validation for the efficiency of our algorithms. These fairness notions could also be extended to the min-knapsack problem. The fair knapsack problems encompass various important problems, such as participatory budgeting, fair budget allocation, and advertising.

KEYWORDS

Fairness; Knapsack; Optimization

1 INTRODUCTION

The knapsack (also known as max knapsack) problem is a classical packing problem. The input of the max-knapsack problem consists of a bounded capacity knapsack and a set of items, each having an associated weight and value. The objective is to select a subset of items such that the total weight of the selected items does not exceed the capacity of the knapsack and the total value is maximized. The min knapsack problem is an extensively studied variant, where the input consists of a set of items, each having an associated weight and value, along with the lower bound on the packing value. The goal of this problem is to select a subset of items such that the total value of the selected items is at least the given bound, and the total weight is minimized. Max knapsack and min knapsack problems represent the prototypical packing and covering problems, respectively.

From the practical viewpoint, the knapsack problem models many prominent industrial problems such as budgeting, cargo packing, resource allocation, assortment planning, etc. [26]. The knapsack problem and its variants are special cases of many salient optimization problems, e.g., the generalized assignment problem, the packing and covering linear programs, etc. They are also key subproblems in the solution of more complex problems, such as the cutting stock problem [44].

In this work, we consider the notion of group fairness in knapsack problems. In this setting, each item belongs to a category, and our goal is to compute a subset of items such that each category is fairly represented, in addition to the total weight of the subset not exceeding the capacity, and the total value of the subset being maximized. We study various notions of group fairness, such as the number of items from each category, the total value of items from each category, and the total weight of items from each category.

In recent years, a considerable amount of research [2, 25, 43] has been focused on group fairness, i.e., to ensure that the algorithms are not biased towards or against any specific group in the population. Here, the two key questions are: how to formalize the notion of fairness and how to design efficient algorithms that conform to these formalization. One such formalization, disparate impact (DI) doctrine [12, 13] posits that any group must have approximately equal or proportional representation in the solution provided by the algorithm. Using this doctrine, Bera et al. [8] introduced a notion of fairness in clustering problems where they deem a solution to be fair if it satisfies two properties: (a) restricted dominance, which upper bounds the fraction of items from a category, and (b) minority protection, which asserts a lower bound on the fraction of items from a category. They use LP based iterative rounding algorithm and the solution obtained by this algorithm might violate the fairness constraints by some additive amount. Similar group fairness notions as studied by Bera et al. [8] for clustering problem, have little been studied for resource allocation problems [7]. Resource allocation problems have been mostly studied under individual fairness and strategic viewpoint [5, 6, 27, 33, 38].

These group fairness notions seem to arise naturally in many practical applications. One such scenario is the case of a server serving multiple clients. These clients can be viewed as categories. The clients have a set of jobs to be processed by the server. Each job has some specific resource requirement, which can be viewed as the weight of the item, and a parameter denoting importance, which can be viewed as the associated value. Since the server has a limited amount of available resources, which can be viewed as the capacity of the knapsack, it has to select a subset of jobs that can be processed using available resources. It is expected that the selected subset is fair for each client.

The knapsack problem with fairness constraints can also be used to model the problem of fair budget allocation in governing bodies. A government may have various project proposals related to different sectors such as agriculture, education, defense, etc. These sectors can be viewed as categories. Each project has some cost and value (indicating social impact), which can be viewed as the associated weight and value of an item, respectively. A government
wants to allocate its budget, which can be viewed as the capacity of the knapsack, such that each sector is fairly addressed and the total social impact is maximized. The fair max-knapsack problem with group value constraint is well suited to model this scenario.

Similarly, we can model the notion of participatory budgeting, which has been proposed recently to take into account the preferences of various stakeholders in the organizations’ budget [3, 20, 41]. In this process, each stakeholder provides a subset of projects, which can be viewed as items, it prefers. The cost of a project could be viewed as the weight of an item. The number of stakeholders preferring a project could be viewed as the value of an item. An organization could be further divided into subparts, which can be viewed as the categories. The stakeholders associated with a particular subpart would have a preference for specific projects. Any budgeting which does not consider such preferences might lead to an allocation that discriminates against some subparts of the organization. The fair knapsack problems described in this work are well suited to model this scenario. There are many other applications of our problems in advertising [21], web search [23], etc.

If all items have identical weights, then the knapsack problem with value lower bound considered in this paper, is a special case of which has been proposed recently to take into account the preferences of various stakeholders in the organizations’ budget [3, 20, 41]. Chekuri and Kumar [11] studied maximum covering problems with group budget constraints. In this problem, we are given a collection of sets \( S = \{S_1, S_2, \ldots, S_m\} \) where each \( S_i \) is a subset of a given ground set \( X \). We are also given a partition of \( S \) into \( l \) groups. The goal is to pick \( k \) sets from \( S \) such that at most one set is selected from each group and the cardinality of their union is maximized. Chekuri and Kumar [11] show that the greedy algorithm gives constant factor approximation to this problem. This problem is analogous to one of our problem that has upper bound on number of items as fairness constraint.

There are also many practical heuristics for the knapsack problems (e.g. see [26, 39]), but they do not consider fairness constraints. It is not straightforward to extend the techniques in these heuristics to the knapsack problem with fairness constraints.

1.1 Our contributions

We study the following three notions of group fairness for the knapsack problem.

- **Bound on the number of items.** Given a set of items, each belonging to one of \( m \) categories, and a range for each category, the problem is to find a subset that maximizes the total value, such that the number of items from each category is in the given range, and the total weight of the subset is at most the capacity of the knapsack.

- **Bound on weight.** Given a set of items, each belonging to one of \( m \) categories, and upper and lower bounds for weight from each category, the problem is to find a subset that maximizes the total value, such that the total weight of items from each category is in between the given bounds, and the total weight of the subset is at most the capacity of the knapsack.

- **Bound on value.** Given a set of items, each belonging to one of \( m \) categories, and upper and lower bounds for value from each category, the problem is to find a subset that maximizes the total value, such that the total value of items from each category is in between the given bounds, and the total weight of the subset is at most the capacity of the knapsack.

### Table 1: Violations by our Algorithms

<table>
<thead>
<tr>
<th>Fairness case</th>
<th>Number of items</th>
<th>Bound on value</th>
<th>Bound on weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>Max</td>
<td>( (1 - \varepsilon, 0, 0) )</td>
<td>((1 - \varepsilon, 0, 0))</td>
<td>((1, \varepsilon))</td>
</tr>
<tr>
<td>Min</td>
<td>((1 + \varepsilon, 0, 0))</td>
<td>((1, \varepsilon))</td>
<td>((1 + \varepsilon, 0))</td>
</tr>
</tbody>
</table>

category is in between the given bounds, and the total weight of the subset is at most the capacity of the knapsack.

We also study similar fairness notions in the min-knapsack problem which are defined analogously. We give approximation algorithms for these problems. For any input error parameter \( \varepsilon > 0 \), the running time of our algorithms is polynomial in the input size and \( \frac{1}{\varepsilon} \). Our algorithms output a solution that having the total value at least \((1 - \varepsilon)\) times the value of the optimal solution (for min-knapsack the weight is at most \((1 + \varepsilon)\) times optimal weight), but the solution produced by some of our algorithms might violate the fairness and/or the knapsack constraints by a small amount (multiplicative factor of \( \varepsilon ) \). For these cases with violations, we show that it is even NP-hard to obtain a feasible solution without violating any constraint. Thus violations are necessary for these cases. We summarize the results of our algorithms in Table 1. First row of the table indicates fairness variants in max-knapsack. Second row of the table indicates fairness variants in min-knapsack. Each entry in the table is represented by a triplet. The first entry in the triplet indicates the approximation ratio achieved by our algorithm. The second entry in the triplet indicates the fractional amount by which the output of the algorithm might violate the fairness constraints. The third entry in the triplet indicates the fractional amount by which the output of the algorithm might violate the knapsack constraint.

Because of the space limitation, we give detailed algorithms for two cases only, bound on value and number of items. The algorithms for other cases could be found in the full version of the paper [36].

**Proof overview:** There is a dynamic programming (DP) based algorithm to solve the classical knapsack problem that runs in pseudo-polynomial time. By rounding the values (or weights) such that the rounded value belongs to a small set (if the problem has \( n \) items to pack, then the value is rounded to the nearest multiple of \( \varepsilon/n \)), the DP technique will give an FPTAS \( \varepsilon \) \([22, 31]\). At a high level, we also use multi-level DP. First, we compute bundles of different values of items from the same category. Next, we select one bundle from each category in divide and conquer fashion using a different DP table. The divide and conquer approach in combining the bundles makes our algorithm practically efficient.

2 RELATED WORK

**Max knapsack problem.** The classical knapsack problem was one of Karp’s 21 NP-complete problems. It is one of the fundamental problems in optimization and approximation algorithms. Knapsack problem admits FPTAS \( \varepsilon \) \([22, 31]\). PTAS \( \varepsilon \) for the multiple knapsack problem is also known \([10, 24]\). The knapsack problem has also

\(^1\)A polynomial-time approximation scheme (PTAS) is an algorithm which takes an instance of an optimization problem and a parameter \( \varepsilon > 0 \) and, in polynomial time of the input size for fixed \( \varepsilon \), produces a solution that is within a factor \( 1 + \varepsilon \) of being
been studied under the multidimensional setting [18, 19]. The max-
knapsack problem when the fairness constraint is an upper bound
on weight is a special case of multidimensional knapsack problem.
Another variant of the knapsack problem is the submodular knap-
sack problem, where the value function is submodular [29, 32, 42].
The knapsack problem has also been well-studied under an on-
line setting [1]. For different variants of knapsack and related bin
packing problems, we refer the readers to [14, 28].

**Min-knapsack problem.** The min-knapsack problem is a frequently
encountered problem in optimization. The problem admits an FP-
TAS [15, 35]. As the problem appears as a key substructure of
numerous other optimization problems, the polyhedral study of
this problem has led to the development of important tools, such as
the knapsack cover inequalities and there is a rich connection of
the problem with extension complexity and sum-of-squares hierarchy
[4, 16, 30].

**Class-constrained knapsack and fair knapsack.** To the best of our
knowledge, the fairness notions for knapsack problems studied in
this work have not been studied previously. The closest model to
our notations, is a PTAS [17]. They have several approaches to aggregate
voters’ preferences: individually best, diverse knapsack and fair
knapsack problem [40]. This problem restricts the maximum
number of classes from which items could be packed in a knapsack.

Let the set \( V = \{1, 2, \ldots, r\} \) be denoted by \([r]\). Let \( m \) denote
the number of categories. The category number is denoted from the
set \([m]\). Each input item belongs to some category. Let \( V_i \) denote
the set of all items from category \( i \), \( \forall i \in [m] \). For \( i \in [m] \), if \( V_i \)
has \( k \) items, then \( V_i = [k] \). Also, assume that \( w_j \) and \( v_j \) are the
weight and the value of item, respectively, \( \forall i \in [m] \). Also let \( \max\{|V_i| | i \in [m]\} = n. \) Let \( N \) be the total number of items. By
our notations, \( N \leq nm \).

Now, we formally define the fair max-knapsack problem with
bound on total value of items from each category. We refer this
problem by the acronym \( BV^{\text{max}} \).

**Problem 1 (\( BV^{\text{max}} \)).** Given a set of items, each belonging to one
of \( m \) categories, a lower bound \( l_i^p \geq 0 \) and an upper bound \( u_i \) such
that \( u_i^p \geq l_i^p \) for each category \( i \), the goal is to find a subset that
maximizes the total value, such that the total value of items from
the category \( i \) is in between \( l_i^p \) and \( u_i \), \( \forall i \in [m] \), and the total weight
of the subset is at most the knapsack capacity \( B \).

We prove that it is even NP-hard to obtain a feasible solution of
an instance of \( BV^{\text{max}} \) (Problem 1).

**Theorem 3.1.** There is no polynomial time algorithm that outputs
a feasible solution of \( BV^{\text{max}} \) (Problem 1), assuming \( P \neq NP \).

**Proof.** Subset sum is a classical NP-hard problem, where we are
given a set \( I \) of non-negative rationals such that \( \sum a_i = 1 \),
and the goal is to find a subset \( S \subseteq I \) such that \( \sum a_i = \frac{1}{2} \). Now
given an instance of subset sum with the set \( I \), we will construct
an instance of \( BV^{\text{max}} \) (Problem 1). Let \( m = 1, l_i = 0, u_i = 1, B = \frac{1}{2} \).
Let \( V_1 \) be the set of items in the instance of \( BV^{\text{max}} \). The items
in \( V_1 \) correspond to the numbers in \( I \). An item corresponding to some
\( a \in I \) has a value and a weight equal to \( a \). This proves that if we have
an algorithm that doesn’t violate the fairness constraint, then we
can solve subset sum. But assuming \( P \neq NP \), this is not possible. \( \square \)

We now give an algorithm for \( BV^{\text{max}} \) (Problem 1) which might
violate fairness constraint for a category by small fraction. The
algorithm uses dynamic programming tables. The entry in the
dynamic programming tables represents the total weight of a subset
of items. The respective subsets can be obtained by maintaining
appropriate references in the tables. Details of it are trivial and
hence they are not discussed in this paper.

**Theorem 3.2.** For any \( \varepsilon > 0 \), there exists an algorithm for \( BV^{\text{max}} \)
(Problem 1) that outputs a set \( S \) having total value at least \( 1 - \varepsilon \) times
the optimal value of \( BV^{\text{max}} \), such that \( (1 - \varepsilon) l_i \leq \sum_{s \in S \cap V_i} \frac{v_{s \cap V_i}}{k} \forall k \in [m] \), and the total weight of \( S \) is at most \( B \). The running
time of the algorithm is \( O\left(n^m \log^3 m \log_{1+\varepsilon} \frac{N_{\text{max}}}{v_{\text{min}}} \right) \), where \( v_{\text{min}} \) :=
min \( \{ v_{j}^{(i)} | i \in [m] \land j \in V_i \land v_{j}^{(i)} > 0 \} \), and \( v_{\text{max}} \) :=
max \( \{ v_{j}^{(i)} | i \in [m] \land j \in V_i \} \).

Towards proving this theorem, we first study the following sub-
problem.

**Problem 2.** Given \( \sigma \geq 0, \varepsilon > 0 \) and a set \( V' = [n] \) of items,
with item \( i \in V' \) having the value \( v_i' \) and the weight \( w_i' \), compute a
subset \( S \subseteq V' \) minimizing \( \sum_{i \in S} w_i' \), such that \( (1 - \varepsilon) \sigma \leq \sum_{i \in S} v_i' \leq (1 + \varepsilon) \sigma \).

Problem 2 looks similar to the min-knapsack problem but it is
different from it in the following way. The total value of output of
min-knapsack problem is required to be more than the given lower
bound, while the total value of the output of Problem 2 is required
to be lying in a small range. We will use Algorithm 1 for Problem 2
to obtain bundles of items in \( V_i, \forall i \in [m] \), such that the total value
of different bundles are in different required ranges.
We can have two cases as below.

1. We give the proof of correctness of the algorithm below. Let $\mathcal{H}(1, \omega'') \in \left[\frac{(1+2\epsilon)n}{\epsilon}\right]$ be the set output by Step 4 of Algorithm 1. By the construction of the dynamic programming table $\mathcal{H}$, we have

$$\min \left\{ \mathcal{H}(n, \omega'') \mid \left[\frac{(1-2\epsilon)n}{\epsilon}\right] \leq \omega'' \leq \left[\frac{(1+2\epsilon)n}{\epsilon}\right] \right\}.$$

Using the definition of $\omega''$ of Step 2, we obtain from the left inequality of (1b),

$$\frac{(1-2\epsilon)n}{\epsilon} - \frac{\epsilon}{n} \leq \sum_{i \in S} q_i' \Rightarrow (1-3\epsilon)n \leq \sum_{i \in S} q_i'. \quad \text{(Definition of } q_i')$$

From right inequality of (1b),

$$\sum_{i \in S} q_i'' + 1 \leq \frac{(1+2\epsilon)n}{\epsilon} + n \Rightarrow \sum_{i \in S} q_i' \leq (1+3\epsilon)n.$$

The last inequality is true by the fact that $q_i''$ is an integer, $\forall i \in V'$. Note that, due to the initialization in Step 3a, the algorithm returns $\infty$ if it does not find a subset in Step 4.

Theorem 3.3. For any $\epsilon > 0$ and $\alpha > 0$, there exists an algorithm for Problem 2 that outputs a subset $S$ having the total weight at most the optimal weight of Problem 2, and $(1-3\epsilon)n \leq \sum_{i \in S} q_i' \leq (1+3\epsilon)n$.

Proof. The algorithm for the theorem is described in Algorithm 1. We give the proof of correctness of the algorithm below.

We claim that the entry $\mathcal{H}(i, \omega'') \in \left[\frac{(1+2\epsilon)n}{\epsilon}\right]$ indicates the weight of the least weight subset from the first $i$ items of $V'$ having the total rounded value equal to $\omega''$. Let $S'$ denote the subset satisfying above property for the entry $\mathcal{H}(i, \omega'')$. We have two cases as below:

- If $i \in S'$, then $\mathcal{H}(i, \omega'')$ is the sum of the weight of $i$ (with $\omega''$), and the weight of minimum weight subset from first $i - 1$ items having the total rounded value $\omega'' - q_i'' (\mathcal{H}(i-1, \omega'' - q_i''))$.

- If $i \notin S'$, then $\mathcal{H}(i, \omega'')$ is equal to the weight of minimum weight subset from first $i$ items having the total rounded value $\omega''$, i.e. $\mathcal{H}(i, \omega'') = \mathcal{H}(i-1, \omega'')$.

The recursion in Step 3b captures both of above possibilities. Step 3a does necessary initialization for the recursion in Step 3b. Let $O$ denote the set of items in an optimal solution of Problem 2 and $S$ be the set output by Step 4 of Algorithm 1. By the construction of the table $\mathcal{H}$, we have

$$\left[\frac{(1-2\epsilon)n}{\epsilon}\right] \leq \sum_{i \in S} q_i'' \leq \left[\frac{(1+2\epsilon)n}{\epsilon}\right].$$

Running time analysis. The size of the table $\mathcal{H}$ is $O\left(\frac{n^2}{\epsilon}\right)$. We require $O(1)$ time to fill each entry. So, the total running time of the algorithm is $O\left(\frac{n^2}{\epsilon}\right)$.

Proof of Theorem 3.2. The algorithm for theorem is described in Algorithm 2. The algorithm creates bundles of items from $V_i$, $\forall i \in [m]$, such that the total value of each bundle is in different ranges using Theorem 3.3 (Step 2). It stores these bundles in the table $\mathcal{W}$. After that the algorithm combines bundles from all categories in divide and conquer fashion to obtain the final solution using the dynamic programming table $X$ (Step 3). The total value of each bundle is represented by some power of $(1 + \epsilon')$ in the tables $\mathcal{W}$ and $X$. So, the algorithm might lose at most $(1 + \epsilon')$ fraction of total value in one iteration of Step 3. The total fraction of value lost in the calculation after combining the bundles from all the categories in Step 3b is at most $(1 + \epsilon')^{O(\log_2 m)}$. This is at most $(1 + \epsilon)$ because of the choice of $\epsilon'$ in Step 1. We describe the formal proof of the algorithm below.

Properties of $\mathcal{W}$. We claim that the entry $\mathcal{W}(i, j), \forall i \in [m], j \in W_{range}$, indicates the weight of the subset of $V_i$ that satisfies the two properties listed below. The entry of $\mathcal{W}$ also corresponds to the respective subset. The entry of $\mathcal{W}$ could be $\infty$, which indicates no subset. We use the notation $W(i, j)$ to indicate both the entry and the subset.
The sets of items: \( S \) having the total value at least \( 1 - \epsilon \) times the optimal value of \( BV_{\max} \), such that
\[
(1-\epsilon)l_i^a \leq \sum_{r \in S \cap V_i} v_{i}^{(r)} \leq (1+\epsilon)u_i^a, \forall i \in [m], \text{and the total weight of } S \text{ is at most } B.
\]

(1) Let \( \epsilon' = \left(1 + \frac{2}{\epsilon}\right) \frac{B_m}{m} \). Also let
\[
W_{\text{range}} := \left\{ \log_2 \left( 1 + \frac{x}{\epsilon} \right) \left( \frac{N_{\max}}{v_{\min}} \right) \right\} \cup \{0\} \text{ and}
\]
\[
X_{\text{range}} := \left\{ \log_2 \left( 1 + \frac{x}{\epsilon} \right) \left( \frac{N_{\max}}{v_{\min}} \right) \right\} \cup \{-\log_2 m, \ldots, -1,0\}.
\]

(2) Let \( W(i,j), \forall i \in [m], \forall j \in W_{\text{range}} \), be the table, where the entry \( W(i,j) \) indicates the weight of a subset of \( V_i \) that is obtained by Theorem 3.3 by setting \( V_i \) as \( V' \), \( v_{\min}(1 + \frac{x}{\epsilon})^j \) as \( v \), and \( \frac{x}{\epsilon} \) as \( \epsilon \) in Theorem 3.3.3.

(3) Let \( X(i,j), \forall i \in [2m-1], \forall j \in X_{\text{range}} \), be the DP table constructed as follows.

(a) \( \forall i \in [m], \forall j \in X_{\text{range}}, \) we have
\[
X(m-1+i,j) := \min \left\{ X(2i,j') + X(2i+1,j') \mid \left( 1 + \frac{x}{\epsilon} \right)^j \geq \left( 1 + \epsilon' \right)^j \right\}
\]
\[
\land \ v_{\min}(1 + \frac{x}{\epsilon})^{j'+1} \geq l_i^a \text{ and } v_{\min}(1 + \frac{x}{\epsilon})^{j'-1} \leq u_i^a \text{ and }
\]
\[
j'' \in W_{\text{range}} \right\}.
\]

If the set satisfying above condition is empty, then set \( X(i,j) \) to \( \infty \).

(b) \( \forall i \in [m-1], \forall j', j'' \in X_{\text{range}}, \) we have
\[
X(i,j) := \min \left\{ X(2i,j') + X(2i+1,j') \mid \left( 1 + \frac{x}{\epsilon} \right)^j \leq \left( 1 + \epsilon' \right)^j \leq \left( 1 + \epsilon' \right)^j \right\}.
\]

If the set satisfying above condition is empty, then set \( X(i,j) \) to \( \infty \).

(4) Output the subset \( S \) in the following way,
\[
\argmax_{j} \left\{ X(i,j) \mid X(i,j) \leq B \right\}.
\]

As a result by slight modification of the proof when \( m \) is not power of 2. The total number of nodes in \( T \) will be \( 2m-1 \). Each node in \( T \) could be represented by an index number from 1 to \( 2m-1 \) with root at index 1. The node at an index \( i \) has a left child at an index \( 2i \) and right child at an index \( 2i+1 \), \( \forall i \in [m-1] \). Let the leaf node at an index \( (m-1) + i \) represent the category \( i, \forall i \in [m] \). Let \( T(i) \) denote the set of categories represented by the leaves of the subtree rooted at \( i \). Specifically, \( T(m-1+i) = \{i\}, \forall i \in [m]\). Also, \( T(i) = T(2i) \cup T(2i+1), \forall i \in [m-1] \).

Properties of \( X \). We claim that the entry \( X(i,j), \forall i \in [2m-1], \forall j \in X_{\text{range}} \), indicates the weight of a subset of \( \cup_{k \in T(i)} V_k \) that satisfies three properties mentioned below. The entry of \( X \) also corresponds to the respective subset. The entry of \( X \) could be \( \infty \), which indicates no subset. We use the notation \( X(i,j) \) to indicate both the entry and the subset.

1. If the entry \( X(i,j) \) is finite, then
\[
\sum_{k \in T(i)} \sum_{r \in X(i,j) \cap V_k} v_{i}^{(k)} \geq \left(1 - \frac{\epsilon}{2}\right) \left(1 + \epsilon'\right)^j v_{\min}.
\]

2. If the entry \( X(i,j) \) is finite, then \( \forall k \in T(i) \),
\[
\left( \frac{1}{2} - \frac{\epsilon}{2} \right) \left(1 + \frac{x}{\epsilon} \right)^j l_k^a \leq \sum_{r \in X(i,j) \cap V_k} v_{i}^{(k)} \leq \left(1 + \frac{x}{\epsilon} \right)^j u_k^a.
\]

3. Let \( i \) be a node of \( T, \forall i \in [2m-1] \), having the distance \( t \) from leaves, \( t \in \left[ \log_2 m \right] \cup \{0\} \). For all \( O' \subseteq \cup_{k \in T(i)} V_k \) having the total value at least \( (1 + \epsilon')^{j+1} v_{\min} \), and \( l_k^a \leq \sum_{r \in O' \cap V_k} v_{i}^{(k)} \forall k \in T(i) \), the total weight of the subset \( X(i,j) \) is at most the total weight of \( O' \).

In Property 3, any considered set \( O' \) will have value at least \( v_{\min} \). Thus in Property 3, \( t+j \geq 0 \), i.e. \( j \geq -\log_2 m \). Hence, the minimum value in \( X_{\text{range}} \) has been set to - \( \log_2 m \) (Step 1).

Steps 3a of the algorithm chooses the subset \( W(i,j') \) that satisfies the inequalities \( (1 + \frac{x}{\epsilon})^{j+1} v_{\min} \geq l_i^a \) and \( (1 + \frac{x}{\epsilon})^{j-1} v_{\min} \leq u_i^a \). By Property 1 of \( W \), the total value of \( W(i,j') \) is in between \( (1 - \frac{\epsilon}{2}) (1 + \frac{x}{\epsilon})^{j+1} l_i^a \) and \( (1 + \frac{x}{\epsilon}) (1 + \frac{x}{\epsilon})^{j-1} u_i^a \). This proves that the subset corresponding to finite entry \( X(i,j), \forall i \in [2m-1], \forall j \in X_{\text{range}}, \) satisfies Property 2.

The subset corresponding to finite entry \( X(m-1+i,j), \forall i \in [m] \) \( \forall j \in X_{\text{range}}, \) will satisfy Property 1. This is true because of Property 1 of \( W \) and the condition \( (1 + \frac{x}{\epsilon})^{j} \geq (1 + \epsilon')^{j} \) in Step 3a for selecting subset \( W(i,j') \). Because of the condition \( (1 + \epsilon')^{j} \leq (1 + \epsilon')^{j'} \) in Step 3b of the algorithm, the subset corresponding to finite entry \( X(i,j), \forall i \in [m-1], \forall j \in X_{\text{range}}, \) will satisfy Property 1.

We prove that the nodes in \( T \) will satisfy Property 3 by induction. In the base case, we prove that Property 3 is satisfied for all leaves. Let \( O' \) be any subset of \( V_i \) that satisfies the fairness bounds such that the total value of \( O' \) is at least \( v_{\min} (1 + \epsilon')^{j} \). Let \( j' \in W_{\text{range}} \) such that the total value of \( O' \) is in between \( v_{\min}(1 + \frac{x}{\epsilon})^{j'+1} \) and \( v_{\min}(1 + \frac{x}{\epsilon})^{j-1} \). So, the total value of \( O'' \) is in between \( v_{\min}(1 + \frac{x}{\epsilon})^{j'} \) and \( v_{\min}(1 + \frac{x}{\epsilon})^{j} \). By Property 2 of \( W \), the total weight of \( W(i,j') \) is at most \( O'' \). Since \( O'' \) satisfies the fairness bounds, the conditions \( v_{\min}(1 + \frac{x}{\epsilon})^{j'+1} \geq l_i^a \) and
\( v_{\min} \left( 1 + \frac{\epsilon}{2} \right)^{j - 1} \leq u_i^v \) in Step 3a are also satisfied. So, the subset ‘\( W(i, j''') \)’ is feasible for \( X(m - 1 + i, j) \) in Step 3a. So, the Property 3 is satisfied by \( X(m - 1 + i, j) \).

For any \( t \in \left[ \log_2 m - 1 \right] \cup \{0\} \), assume the hypothesis that all the nodes having distance \( t \) from leaves satisfy Property 3. We will prove by induction that for any node \( i \) with distance \( t+1 \) from leaves, the node \( i \) satisfy Property 3. Let \( O' \subseteq \bigcup_{i \in T(i)} V_i \) that satisfies the fairness conditions for all categories in \( T(i) \). Let \( j' \in X_{range} \) that satisfies the following inequality

\[
(1 + \epsilon')^{j'} v_{\min} \leq \sum_{k \in T(2i)} \sum_{r \in \mathcal{R}'(k)} v_r(k) = (1 + \epsilon')^{j'+1} v_{\min}.
\]  

If \( j' = j^* - t \), the induction hypothesis implies the following:

\[
\sum_{k \in T(2i)} \sum_{r \in \mathcal{R}'(k)} w_r(k) \leq \sum_{k \in T(2i)} \sum_{r \in \mathcal{R}'(k)} w_r(k).
\]

Similarly, let \( j'' \in X_{range} \) that satisfies the following inequality

\[
(1 + \epsilon')^{j''} v_{\min} \leq \sum_{k \in T(2i+1)} \sum_{r \in \mathcal{R}'(k)} v_r(k) \leq (1 + \epsilon')^{j''+1} v_{\min}.
\]

If \( j'' = j^* - t \), the induction hypothesis implies the following

\[
\sum_{k \in T(2i+1)} \sum_{r \in \mathcal{R}'(k)} w_r(k) \leq \sum_{k \in T(2i+1)} \sum_{r \in \mathcal{R}'(k)} w_r(k).
\]

We claim that the inequality \( (1 + \epsilon')^{j'} \leq (1 + \epsilon')^{j''} + (1 + \epsilon')^{j''+1} \) is satisfied in Step 3b for any \( j \in X_{range} \), such that the total value of \( O' \) is at least \( v_{\min}(1 + \epsilon')^{j''+1} \). This is true because the maximum value of \( O' \) is at most \( v_{\min}(1 + \epsilon')^{j''+1} \) (by Inequality 2 and Inequality 4), which is more than \( v_{\min}(1 + \epsilon')^{j''+1} \). So, the pair of subsets \( X(2i, j') \) and \( X(2i+1, j'') \) is feasible in Step 3b for all such \( j \). By Equation 3 and Equation 5, Property 3 of \( X \) is satisfied for \( (i, j) \) and \( O' \).

Let \( OPT \) be the optimal value. Let \( j \) be the number in \( X_{range} \) such that

\[
(1 + \epsilon')^{j + \log_2 m} v_{\min} \leq OPT \leq (1 + \epsilon')^{j + \log_2 m+1} v_{\min}.
\]

If we apply Property 3 of \( X \) to any optimal solution, we get that the total weight of the subset \( X(1, j) \) is less than or equal to the total weight of an optimal subset. So, the subset \( X(1, j) \) is feasible in Step 4. By Property 1, the total value of \( X(1, j) \) is at least \( \left( 1 + \frac{\epsilon}{2} \right)^j v_{\min} \). By Inequality 6, this is at least

\[
\left( \frac{1 - \epsilon}{2} \right)^{OPT} (1 + \epsilon')^{j + \log_2 m+1} v_{\min} \geq \left( \frac{1 - \epsilon}{2} \right)^{OPT} (1 + \frac{3\epsilon}{8}) v_{\min}.
\]

If the entry \( X(1, j) \) is finite, Property 2 of \( X \) implies that the total value of items from \( V_i \cap X(1, j) \) is between \( (1 - \frac{\epsilon}{2})^j (1 + \frac{\epsilon}{8}) u_i^v \) and \( (1 + \frac{\epsilon}{2}) (1 + \frac{\epsilon}{8}) u_i^v \) for all \( i \in [m] \). The total value of items from \( V_i \cap X(1, j) \) is at least \( (1 - \frac{\epsilon}{2})^j (1 + \frac{\epsilon}{8}) u_i^v \)


\( (1 - \epsilon) u_i^v \). The total value of items from \( V_i \cap X(1, j) \) is at most \( (1 + \frac{\epsilon}{2}) (1 + \frac{\epsilon}{8}) u_i^v < (1 + \frac{\epsilon}{8}) u_i^v < (1 + \epsilon) u_i^v \).


Running time analysis: The size of the \( W \) is \( O \left( m \log_{1+\epsilon} \left( \frac{N_{\max}}{v_{\min}} \right) \right) \). We require \( O \left( \frac{n^2}{\epsilon} \right) \) time to fill each entry of the \( W \). So, the total time required to build the table ‘\( W \)’. So, the total running time of the algorithm is \( O \left( m \log_3^2 m \log_{1+\epsilon} \left( \frac{N_{\max}}{v_{\min}} \right) \right) \). So, the total running time of the algorithm is \( O \left( m \log_3^2 m \log_{1+\epsilon} \left( \frac{N_{\max}}{v_{\min}} \right) \right) \).

4 ALGORITHM FOR BOUND ON NUMBER OF ITEMS IN MAX-KNAPSACK

The notations used in this section are same as in Section 3. We formally define the fair max-knapsack problem with bound on the number of items from each category. We refer this problem by the acronym BN_{max}.

Problem 3 (BN_{max}). Given a set of items, each belonging to one of \( m \) categories, and the numbers \( l_i^v \) and \( u_i^v \) for each category \( i, v \in [m] \), the problem is to find a subset that maximizes the total value, such that the number of items from category \( i \) between \( l_i^v \) and \( u_i^v \), \( v \in [m] \), and the total weight of the subset is at most the capacity of the knapsack \( B \).

Theorem 4.1. For any \( \epsilon > 0 \), there exists a \( (1 - \epsilon) \)-approximation algorithm for BN_{max} (Problem 3) with running time \( O \left( \frac{N_v}{\epsilon} \right) \).

Proof. The algorithm is described in Algorithm 3. The algorithm starts by rounding the values of all items so that the rounded values lie in a small range (Step 2). Then it creates bundles of different cardinality and total rounded value from items in \( V_i, v \in [m] \). It uses the dynamic programming table \( A(i, j, v, t) \) for this (Step 3). After that the algorithm combines bundles from all categories to obtain the final solution using the dynamic programming table \( B \) (Step 4). The following describes formal proof of the algorithm.

Property of \( A \). We claim that \( A(i, j, v, t), v \in [m], t \in V_i \cup \{0\}, j \in V_i, \forall v \in \left[ \frac{N_v}{t} \right] \cup \{0\} \), denotes the weight of a minimum weight subset of cardinality \( t \) from first \( j \) items of \( V_i \) having the total rounded value \( v \). Let \( S' \) be the subset satisfying above property for the entry \( A(i, j, v, t) \).

- If \( j \in S' \), then \( A(i, j, v, t) \) is the sum of weight of \( j \) and the weight of minimum weight subset from first \( j - 1 \) items of \( V_i \) having cardinality \( t - 1 \) and the rounded value \( v - v^{(j)} \), which is equal to \( A(i, j - 1, v - v^{(j)}), t - 1 \) \( + w_j \).
- If \( j \notin S' \), then \( A(i, j, v, t) \) is equal to the weight of minimum weight subset from first \( j - 1 \) items of \( V_i \) having cardinality \( t \) and the rounded value \( v \), which is equal to \( A(i, j - 1, v, t) \).

The recursion in Step 3b captures both of above possibilities. The Step 3a initializes the base cases for the recursion in Step 3b. The entry of \( A \) also corresponds to the respective subset. The entry
\textbf{Algorithm 3: Algorithm for }BN^{\max}\textbf{ (Problem 3)}

\textbf{Input:} The sets \( V_i \) of items, \( 0 \leq I_i \leq u_i \) for \( i \in [m] \), \( B \) the capacity of knapsack and \( \varepsilon > 0 \).

\textbf{Output:} \( S \) having the total value at least \( 1 - \varepsilon \) times the optimal value of \( BN^{\max} \) (Problem 3), such that \( I_i \leq S \cap V_i \leq u_i \), \( \forall i \in [m] \) and the total weight of \( S \) is at most \( B \).

(1) Remove all items \( j \in V_i, \forall i \in [m] \) that do not come under any feasible solution. We can check whether an item comes under a feasible solution by checking the weight of least weight feasible subset containing the item is less than or equal to \( B \). Let \( m_{\max} \) be the maximum value of the remaining items.

(2) \( \forall i \in [m], \forall j \in V_i \), let

\[ v^{(i)}_j := \frac{v^{(i)}_j}{\varepsilon v_{\max}}. \]

(3) Let \( \mathcal{A}(i, j, v, t), \forall i \in [m], \forall j \in V_i, \forall t \in V_i \cup \{0\}, \forall v \in \left[ \frac{N}{\varepsilon} \right] \cup \{0\} \), be the dynamic programming table constructed in the following way.

(a) \( \forall i \in [m], \)

\[ \mathcal{A}(i, 1, i_0, t) := v^{(i)}_1, \mathcal{A}(i, 1, 0, 0) = 0. \]

\[ \mathcal{A}(i, 1, i', t) := \infty \text{ for } i' \in \left[ \frac{N}{\varepsilon} \right] \backslash \{v^{(i)}_j\}. \]

(b) \( \forall i \in [m], \forall j \in V_i \backslash \{1\}, \forall t \in V_i \cup \{0\}, \forall v \in \left[ \frac{N}{\varepsilon} \right] \cup \{0\} \),

\[ \mathcal{A}(i, j, v, t) := \min \left\{ \mathcal{A}(i, j-1, v-v^{(i)}_j, t-1) + v^{(i)}_j, \right. \]

\[ \mathcal{A}(i, j-1, v, t) \mid 0 \leq v^{(i)}_j \leq v \} . \]

(4) Let \( \mathcal{B}(i, v), \forall i \in [m], \forall v \in \left[ \frac{N}{\varepsilon} \right] \cup \{0\} \) be another dynamic programming table.

(a) \( \forall v \in \left[ \frac{N}{\varepsilon} \right] \cup \{0\} \),

\[ \mathcal{B}(1, v) := \min \left\{ \mathcal{A}(1, | V_1 \cup v, t) \mid I_1 \leq t \leq u_1 \right\} . \]

(b) For \( i \in [m] \backslash \{1\} \) and \( v \in \left[ \frac{N}{\varepsilon} \right] \cup \{0\} \),

\[ \mathcal{B}(i, v) := \min \left\{ \mathcal{B}(i-1, v), \mathcal{A}(i, | V_i \cup v, t) \mid \right. \]

\[ I_i \leq t \leq u_i \cup v \cup v \in \left[ \frac{N}{\varepsilon} \right] \cup \{0\} \} . \]

(5) Output \( S \) in the following way,

\[ \arg \max_{\mathcal{A}} \left\{ \mathcal{B}(m, v) \mid \mathcal{B}(m, v) \leq B \right\} . \]

\begin{itemize}
  \item of \( \mathcal{A} \) could be \( \infty \), which indicates no subset. We use the notation \( \mathcal{A}(i, j, v, t) \) to indicate both the entry and the subset.
  \item Property of \( \mathcal{B}. \) We claim that \( \mathcal{B}(i, v), \forall i \in [m], \forall v \in \left[ \frac{N}{\varepsilon} \right] \cup \{0\} \), denotes the weight of a minimum weight subset of \( \cup_{j=1}^{i} V_j \) having the total rounded value \( v \), such that fairness constraints are satisfies for the subset for all the categories up to \( i \). Let \( S' \) be the subset of \( \cup_{j=1}^{i} V_j \) satisfying above property for \( B(i, v) \).
  \item Let \( \sum_{j \in S \cap V_i} v^{(i)}_j = \gamma \), \( B(i, v) \) is the sum of weights of minimum weight subset of \( \cup_{j=1}^{i} V_j \) having the total rounded value \( v \), that satisfies fairness condition for all categories up to \( i \) \( \{ B(i-1, v), \gamma \} \), and the minimum weight subset of \( V_i \) having total rounded value \( v \), that satisfies fairness condition for category \( i \) \( \{ A(i, | V_i \cup v, 0, t) \} \), \( t \) such that \( I_i \leq t \leq u_i \). The recursion in Step 4b captures this for all possible \( v \). The entry of \( \mathcal{B} \) also corresponds to the respective subset. The entry of \( \mathcal{B} \) could be \( \infty \), which indicates no subset. We use the notation \( \mathcal{B}(i, v) \) to indicate both the entry and the subset.
  \item By the property of \( \mathcal{B} \), the total rounded value of \( S \) (Step 5) is at least the total rounded value of any optimal solution. By Lemma 4.2, we can show that the total value of \( S \) is at least \( 1 - \varepsilon \) times the optimal solution.
\end{itemize}

\textbf{Running time analysis:} The size of the table \( \mathcal{A} \) is \( O \left( \frac{mN^2}{\varepsilon} \right) \), and to fill each entry in the table \( \mathcal{A} \), we require \( O(1) \) time. The size of the table \( \mathcal{B} \) is \( O \left( \frac{mN^2}{\varepsilon} \right) \), and to fill each entry in the table \( \mathcal{B} \), we require \( O \left( \frac{mN^2}{\varepsilon} \right) \) time. So, the total time required is \( O \left( \frac{mN^2}{\varepsilon} \right) \).

\textbf{Lemma 4.2.} If \( OPT \) is the optimal objective value of \( BN^{\max} \) (Problem 3), then the total value of a set returned by Algorithm 3 is at least \( 1 - \varepsilon \)OPT.

\textbf{Proof.} Let \( O \subseteq \cup_{i=1}^{m} V_i \) be the set of items in an optimal solution and \( S \subseteq \cup_{i=1}^{m} V_i \) be the set of items selected by Algorithm 3. Since \( S \) is an optimal solution of rounded value in Algorithm 3, the total rounded value of \( S \) is more than the total rounded value of \( O \).

\[ \sum_{i=1}^{m} \sum_{j \in V_i \cap O} v^{(i)}_j \leq \sum_{i=1}^{m} \sum_{j \in V_i \cap S} v^{(i)}_j. \] (7)

Because of the rounding in Step 2, we have following inequalities \( \forall i \in [m] \) and \( \forall v \in V_i \),

\[ \frac{\varepsilon v_{\max} v^{(i)}_j}{N} \leq v^{(i)}_j \leq \frac{\varepsilon v_{\max}}{N} v^{(i)}_j + 1. \] (8)

So we get,

\[ \sum_{i=1}^{m} \sum_{j \in V_i \cap O} v^{(i)}_j \leq \frac{\varepsilon v_{\max}}{N} \left( \sum_{i=1}^{m} \sum_{j \in V_i \cap O} v^{(i)}_j + 1 \right) \] (From Ineq. 8)

\[ \leq \frac{\varepsilon v_{\max}}{N} \left( \sum_{i=1}^{m} \sum_{j \in V_i \cap S} v^{(i)}_j \right) + \varepsilon v_{\max} \] (Ineq. 7)

\[ \leq \left( \sum_{i=1}^{m} \sum_{j \in V_i \cap S} v^{(i)}_j \right) + \varepsilon v_{\max}. \] (Ineq. 8)

Since Step 1 of the Algorithm 3 discards all the items which doesn’t come in any feasible solution, we know that \( v_{\max} \leq OPT \).
Therefore, we get
\[
\left( \sum_{i=1}^{m} \sum_{j \in V_i \cup IS} v^{(i)}_{j} \right) \geq (1-\epsilon)OPT.
\]

\[\square\]

5 OTHER PROBLEMS

In this section, we formally define other fairness models that we consider. We give detailed algorithms and hardness results for these problems in the supplementary material.

- Bound on weight in max-knapsack: We refer this problem by the acronym BW\textsuperscript{max}.

**Problem 4 (BW\textsuperscript{max}).** Given a set of items, each belonging to one of \(m\) categories, a lower bound \(l^w_i\) and an upper bound \(u^w_v\) for each category \(i, \forall i \in [m]\), the goal is to find a subset that maximizes the total value, such that the total weight of items from category \(i\) is in between \([l^w_i, u^w_v]\), \(\forall i \in [m]\) and the total weight of the subset is at most the capacity of the knapsack \(B\).

- Bound on number of items in min-knapsack: We refer this problem by the acronym BN\textsuperscript{min}.

**Problem 5 (BN\textsuperscript{min}).** Given a set of items, each belonging to one of \(m\) categories, and numbers \(l^v_i\) and \(u^v_i\) for category \(i\), \(\forall i \in [m]\), the problem is to find a subset that minimizes the total weight, such that the number of items from category \(i\) is between \([l^v_i, u^v_i]\), \(\forall i \in [m]\), and the total value of the subset is at least the given value lower bound \(L\).

- Bound on value in min-knapsack: We refer this problem by the acronym BV\textsuperscript{min}.

**Problem 6 (BV\textsuperscript{min}).** Given a set of items, each belonging to one of \(m\) categories, and numbers \(l^v_i\) and \(u^v_i\) for category \(i\), \(\forall i \in [m]\), the problem is to find a subset that minimizes the total weight, such that the total value of items from category \(i\) is between \([l^v_i, u^v_i]\), \(\forall i \in [m]\), and the total value of the subset is at least the given value lower bound \(L\).

- Bound on weight in min-knapsack: We refer this problem by the acronym BW\textsuperscript{min}.

**Problem 7 (BW\textsuperscript{min}).** Given a set of items, each belonging to one of \(m\) categories, and a lower bound \(l^v_i\) and an upper bound \(u^v_i\) such that \(0 \leq l^v_i \leq u^v_i\) for each category \(i \in [m]\), the goal is to find a subset that minimizes the total weight, such that the total weight of items from each category \(i\) is in between the given bounds \([l^v_i, u^v_i]\), \(\forall i \in [m]\), and the total value of the subset is at least the given bound \(L\).

6 EXPERIMENTAL RESULTS

Our theorems give theoretical guarantees for the quality of the solution output by our algorithms. In this section, we present an experimental evaluation of the running time of Algorithm 2. We experiment\footnote{The code is available on https://github.com/creativolve/GroupFairness_Knapsack.} with Algorithm 2 on randomly generated instances for Problem 1. We note that [26] also uses similar model of random instances for experimental results for the standard knapsack problems.

We give experimental results for the implementation of Algorithm 2 in Table 2. The code was parallelized wherever possible to improve the running time. The model of GPU used by the code was GeForce GTX 1080 Ti. It has 3584 cores. The processor of host system was Intel(R) Xeon(R) Silver 4110 CPU, 2.10GHz.

The input instance is a randomly generated instance. The weight and value of each item is a randomly generated integer between 1 to 100. In all the instances, the number of items in each category is 157. So the total number of items \(N\) is 10048, 5024, and 2512, for \(m = 64, 32, 16\), respectively. The knapsack weight is set to \(\frac{1}{3}\) of expected total weight of items in each case. The lower bounds of categories are set to \(\frac{1}{4}\) of expected total value of items in a category in each case. The upper bounds of categories are set to be the expected total value of items in a category in each case. We report our experimental results in Table 2. For each combination of \(m\) and epsilon, we report the average of running time (in minutes) over 10 different randomly generated input instances.

7 CONCLUSION

In this paper, we studied various fairness notions for knapsack problems. We studied six variants, three for each of max-knapsack and min-knapsack. These different variants encompass several interesting problems. Studying fairness notions in related problems such as multiple knapsack problem [24], multidimensional knapsack problem [18], submodular knapsack problem [32], is an interesting open problem. Study of different fairness notions for resource allocation and scheduling problems remains an interesting area.

ACKNOWLEDGMENTS

AI was supported in part by SERB Award ECR/2017/003296 and a Pratiksha Trust Young Investigator Award.

REFERENCES


[38] François Vanderbeck. 1999. Computational study of a column generation algo-