Weighted Set Multi-Cover on Bounded Universe and Applications in Package Recommendation*

NIMA SHAHBAZI, Department of Computer Science, University of Illinois Chicago, USA
ARYAN ESMAILPOUR, Department of Computer Science, University of Illinois Chicago, USA
STAVROS SINTOS, Department of Computer Science, University of Illinois Chicago, USA

The weighted set multi-cover problem is a fundamental generalization of set cover that arises in data-driven applications where one must select a small, low-cost subset from a large collection of candidates under coverage constraints. In data management settings, such problems arise naturally either as expressive database queries or as post-processing steps over query results, for example, when selecting representative or diverse subsets from large relations returned by database queries for decision support, recommendation, fairness-aware data selection, or crowd-sourcing. While the general weighted set multi-cover problem is NP-complete, many practical workloads involve a *bounded universe* of attributes or items that must be covered, leading to the Weighted Set Multi-Cover with Bounded Universe (WSMC-BU) problem, where the universe size is constant. Despite its relevance in large-scale data processing pipelines, little is known about efficient algorithms for this special case. In this paper, we develop exact and approximation algorithms for WSMC-BU. We first discuss a dynamic programming algorithm that solves WSMC-BU exactly in $O(n^{\ell+1})$ time, where n is the number of input sets and $\ell = O(1)$ is the universe size. We then present a 2-approximation algorithm based on linear programming and rounding, running in $O(\mathcal{L}(n))$ time, where $\mathcal{L}(n)$ denotes the complexity of solving a linear program with $O(n)$ variables. To further improve efficiency for large datasets, we propose a faster $(2+\varepsilon)$-approximation algorithm with running time $O(n \log n + \mathcal{L}(\log W))$, where W is the ratio of the total weight to the minimum weight, and ε is an arbitrary constant specified by the user. Our results provide the first practical constant-factor approximation algorithms for WSMC-BU, with approximation guarantees independent of the universe size. Extensive experiments on real and synthetic datasets demonstrate that our methods consistently outperform greedy and standard LP-rounding baselines in both solution quality and runtime, making them suitable for data-intensive selection tasks over large query outputs.

CCS Concepts: • **Theory of computation → Packing and covering problems**.

Additional Key Words and Phrases: Weighted Set Multi-cover, Set Cover, Bounded Universe, LP Rounding

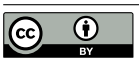
ACM Reference Format:
Nima Shahbazi, Aryan Esmailpour, and Stavros Sintos. 2026. Weighted Set Multi-Cover on Bounded Universe and Applications in Package Recommendation. *Proc. ACM Manag. Data* 4, 3 (SIGMOD), Article 249 (June 2026), 25 pages. https://doi.org/10.1145/3802126

1 Introduction

Set cover is one of the most fundamental problems in computer science, with applications across diverse domains such as network design, information retrieval, resource allocation, sensor networks, VLSI design, machine learning, crew scheduling, facility location, computational biology, and

*This work was partially supported by NSF grant IIS-2348919.

Authors' Contact Information: Nima Shahbazi, Department of Computer Science, University of Illinois Chicago, Chicago, Illinois, USA, nshahb3@uic.edu; Aryan Esmailpour, Department of Computer Science, University of Illinois Chicago, Chicago, Illinois, USA, aesmai2@uic.edu; Stavros Sintos, Department of Computer Science, University of Illinois Chicago, Chicago, Illinois, USA, stavros@uic.edu.



ACM 2836-6573/2026/6-ART249
https://doi.org/10.1145/3802126

network security [14, 25, 27, 37, 48, 49, 60]. In databases, the set cover problem models several data management and query processing tasks, including pipelined filter optimization [40], query routing in big data architectures [41], and package-to-group recommendations [44, 50]. Formally, given a universe $\mathcal{G}$ of ℓ items and a family $\mathcal{D}$ of n subsets of $\mathcal{G}$, the goal is to select the smallest number of sets from $\mathcal{D}$ whose union covers all items in $\mathcal{G}$. Despite being one of Karp's 21 NP-complete problems, both a greedy algorithm and an LP-based rounding algorithm achieve $O(\log \ell)$-approximation [15], which is known to be optimal [18].

Numerous variants of the set cover problem have been studied in both the theory and database communities. Examples include the *set multi-cover problem* [19] the *multi-set multi-cover problem* [28] and the *fair set cover problem* [17] . For all these variants, weighted versions have also been investigated, where each set is associated with a weight and the objective is to minimize the total weight of the selected sets while satisfying the covering constraints.

In this work, we focus on the *weighted set multi-cover problem*. More formally, let $\mathcal{G}$ be a universe of ℓ items where each item $g \in \mathcal{G}$ has a non-negative demand Q_g, and let $\mathcal{D}$ be a family of n subsets of $\mathcal{G}$, where each set $t \in \mathcal{D}$ has an associated non-negative weight w_t. The goal is to select a collection of sets from $\mathcal{D}$ with minimum total weight such that every item $g \in \mathcal{G}$ is covered at least Q_g times. This problem is NP-complete, yet, interestingly, it also admits a greedy $O(\log \ell)$-approximation algorithm [19], where in each iteration it adds in the solution the set minimizing the ratio of its weight over the number of items with positive demands it covers. Furthermore, the standard LP-rounding algorithm for the weighted set multi-cover problem returns an $O(\log \ell)$-approximation solution with high probability [59].

In data management settings, the weighted set multi-cover problem captures the algorithmic core of a class of expressive selection tasks over large relations. Conceptually, these tasks can be viewed as *package recommendation* or *constrained selection queries* [5, 6, 21], where the goal is to retrieve a package (subset) of tuples that satisfies global coverage or representation constraints over attributes, while minimizing a cost function, such as total weight, resource usage, or acquisition cost. Such constraints go beyond what is directly expressible in standard SQL, and therefore are often handled either through specialized query operators or as post-processing steps over query results in data analytics pipelines.

The weighted set multi-cover problem has a wide range of applications in data management, decision support, and data selection. To demonstrate this, we next describe some representative data management settings where this problem arises naturally.

Fairness-aware data selection: [22, 42, 53, 57] A common problem in data management and ML pipelines is to construct a training dataset that satisfies specified representation requirements over demographic groups.

***Example* 1:** In the Chameleon system [22], they are given a dataset of face images, where each image belongs to one or more demographic groups (e.g., "female", "Black", "young"). To ensure that the downstream model is not biased, someone must select sufficiently many images from each demographic group. Since an image can simultaneously belong to multiple groups (e.g., an image may be both "female" and "young"), every selected image contributes coverage to multiple demographic requirements. The goal is to compute a small number of images to satisfy all requirements.

Similar problems appear in many other fairness applications. For instance, consider student admissions in a university, where the goal is to select a cohort of applicants while minimizing the total financial aid or scholarship budget. Each applicant is characterized by protected attributes such as gender and ethnicity, and institutions are often subject to diversity constraints. For instance, a Hispanic-Serving Institution in the U.S. may be required to admit at least 25% Hispanic students, and, due to persistent gender imbalance in STEM fields, a university may additionally require

that at least 25% of admitted students be women. In this formulation, each applicant corresponds to a set with an associated cost (the offered financial aid), and each protected attribute category corresponds to an item with a specified demand.

Our techniques are applicable in all these examples, where the goal is to select a subset of data to satisfy selection requirements across different demographics.

Crowd-Sourced Task Assignment / Data Enrichment: [35, 65, 66] Generally, the problem can be formulated as follows: standard DB work on task assignment and crowdsourcing frames the problem as selecting workers (each with a cost and set of skills) to cover a set of tasks. This is exactly an instance of the weighted set multi-cover problem: Each worker defines a set that covers the skill/task types they can do, each skill/task represents an item in the universe, each demand is the number of worker-assignments needed per task/type, and the weight is defined as the worker cost. To highlight it further, we show the following example, with an overview shown in Figure 1.

***Example* 2:** In crowd-sourced data enrichment, the data is often collected from unreliable sources on the internet and needs verification using a human in the loop [23]. For this, companies often use expert pools like *Amazon Mechanical Turk* to access specialized experts who possess varying proficiencies in domains such as medicine, law, and finance [31]. Each expert has an hourly rate, and the goal is to systematically verify specific attributes—such as the biosafety level of a research lab or the regulatory compliance of a financial institution- to ensure they are accurate before they are finalized in a knowledge graph [20]. Since these attributes often carry high stakes, the task requires a predefined level of redundancy where multiple independent experts must reach a consensus on the same data point to establish it as ground truth [54]. For example, a task may pose the specific question, "Does the provided documentation confirm that this medical facility is licensed for radiology?" This question is distributed to a set of multiple experts, and their responses are aggregated to ensure reliability; the answer is only integrated into the database if a strict consensus is met, such as through a simple majority vote or a requirement that at least 90% of the experts agree on the response. This process involves identifying the required expertise for each attribute, evaluating the available solver pool for overlapping skill sets that can satisfy several domain requirements simultaneously, and selecting the most cost-effective group of individuals to meet the total redundancy constraints [7, 24]. The final goal of this pipeline is the production of an enriched knowledge graph, which serves as a highly reliable and structured database where raw, noisy information has been transformed into a network of verified entities and relationships. An overview of this pipeline is shown in Figure 1.

Constrained package recommendations: [44, 50] Given a user's preferences, the goal is to recommend a package of items that minimizes cost while satisfying the specified constraints.

***Example* 3:** Consider a recommendation application that has access to a dataset containing a collection of restaurants (e.g., the Yelp dataset [29]). Suppose a user specifies the following cuisine preferences: *very interested* in Italian and omnivore cuisines, *interested* in Japanese cuisine, and *somewhat interested* in Mediterranean and vegan cuisines. Each restaurant may offer multiple cuisines; for example, *Orsa & Winston* in Los Angeles serves Japanese, Italian, and omnivore dishes. In addition, each restaurant is associated with an average price per person. The objective is to select a set of restaurants with a minimum total cost that includes at least 3 Italian, 2 Japanese, 1 Mediterranean, 3 omnivore, and 1 vegan restaurants.

***Example* 4:** Consider a software company that must hire developers with experience in multiple programming languages. Specifically, the company requires at least 10 developers with experience in Python, 7 with experience in C, 5 with experience in Java, and at least 3 with experience in PHP. Each developer lists the programming languages they are proficient in, along with an

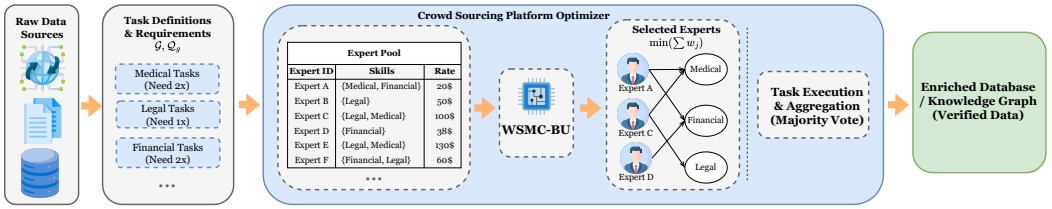


Fig. 1. Overview of crowd-sourced data enrichment using WSMC-BU.

associated salary. A developer who is proficient in multiple languages can simultaneously satisfy the requirements for each of those languages (for example, a developer proficient in both C and Python can count toward both the C and Python requirements). The objective is to select a set of developers with a minimum total salary that satisfies all the language requirements.

All the examples above are instances of the weighted set multi-cover problem. Since this problem is NP-complete, in the above applications, the existing literature often relies on a greedy approach to solve the problem [22, 50]. This is due to the simplicity of the greedy algorithm, while offering a reasonable approximation. Indeed, it is known that the approximation factor of the greedy algorithm cannot be improved in the general settings (unbounded universe) [19]. However, we observe that a common feature of all these applications is that while the number of sets is large (e.g., images, students, restaurants, experts), the universe size $\ell = |\mathcal{G}|$ is relatively small (e.g., number of skills, tasks, demographic groups, or cuisines). Hence, it is reasonable to assume that in many data management and package recommendation tasks, the universe size is bounded by a constant, i.e., $\ell = O(1)$. This motivates the study of the *Weighted Set Multi-Cover problem on Bounded Universe*, or WSMC-BU for short. Its definition is identical to the weighted set multi-cover problem, except that $|\mathcal{G}| = O(1)$.

Despite the many applications of WSMC-BU, research on this special case remains limited. Bredereck et al. [4], showed that the weighted set multi-cover problem is fixed parameter tractable, when parameterized by the size of the universe, implying that WSMC-BU is not NP-hard. However, while being theoretically significant, their approach is of limited practical relevance: the exact polynomials are not analyzed, and polynomial exponents appear to be significantly large, and to the best of our knowledge, no implementation exists due to its complexity. This motivates the search for simpler exact algorithms and for efficient approximation algorithms. Recall that for the weighted set multi-cover problem (unbounded universe), the greedy algorithm achieves an $O(\log \ell)$-approximation. This directly implies an $O(1)$-approximation for WSMC-BU in $O(n \log n)$ time. Similarly, the standard LP-rounding algorithm achieves an $O(\log \ell)$-approximation with high probability, but it is outperformed by the greedy algorithm in the running time.[1] Beyond these results, no approximation algorithms are specifically known for WSMC-BU.

This leaves several fundamental open questions: Is there a simple exact algorithm for WSMC-BU that runs in polynomial time and is practical to implement? Is there a practical approximation algorithm with theoretical guarantees that outperforms the greedy algorithm both in theory and in practice? More specifically, is there an efficient truly constant-factor approximation for WSMC-BU whose guarantee is independent of the input parameters ℓ and n? Can such an algorithm also outperform the greedy and the standard LP-rounding method in practice, both in efficacy (total weight of the returned sets) and efficiency (runtime)? We answer all of these questions affirmatively.

[1]The standard LP formulation for the weighted set multi-cover problem, shown in LP (15), serves as a baseline for our experiments.

Our contributions. Our contributions are summarized as follows.

- In Section 3, we discuss exact polynomial time algorithms for the WSMC-BU problem. As a warm-up, we show a simple exact Dynamic Programming algorithm that runs in $O(n^{\ell+1})$ time.
- In Section 4, we study approximation algorithms for WSMC-BU. First, in Section 4.1 we design a 2-approximation algorithm that runs in $O(\mathcal{L}(n))$ time, where $\mathcal{L}(n)$ is the time to solve an LP with n variables and $O(n)$ constraints.[2].
  Next, in Section 4.2, we modify our previous approximation algorithm to improve its running time. For any given constant $\varepsilon > 0$, we design a $(2+\varepsilon)$-approximation algorithm in $O(n \log n + \mathcal{L}(\log W))$ time, where $W = \frac{\sum_{t\in\mathcal{D}} w_t}{\min_{t\in\mathcal{D}} w_t}$.
- In Section 5, we implement our algorithms and compare them against the Greedy and the standard LP-rounding baselines on real and synthetic datasets. Our approximation algorithm consistently produces solutions of smaller weight than baselines: in real datasets, baselines' solutions are up to 1.3 times larger, and in synthetic datasets up to 2 times larger. Moreover, the $(2+\varepsilon)$-approximation algorithm typically runs faster than the LP baseline, and competes with the greedy algorithm while being faster in many cases.

An additional desirable property in applications is to avoid *over-satisfying* the demands of individual items. That is, while every demand must be met, selecting sets that exceed the required coverage by a large margin may lead to "overfitting" on certain items, thereby wasting resources or biasing the solution towards a few categories. The notion of oversatisfying demands refers to situations where, in order to satisfy the demands of certain groups, the solution is forced to include many elements belonging to another group whose demand is low. This is not directly controlled by a boundary on the number of items but is rather a consequence of the structure of the sets. Ideally, algorithms should satisfy each demand closely, without introducing much unnecessary coverage. As we show in our experimental evaluation (Section 5), our algorithms naturally achieve this property, producing solutions that satisfy demands without significantly overshooting them, in contrast to the greedy baseline, which often overcovers certain items.

2 Preliminaries

2.1 Problem definition

First, we formally define the Weighted Set Multi-Cover problem with Bounded Universe (WSMC-BU) and give some useful definitions that we will use throughout the paper.

Definition 1 (Weighted Set Multi-Cover problem with Bounded Universe — WSMC-BU). *Let $\mathcal{G}$ be a universe of $\ell = O(1)$ items, where each item $g \in \mathcal{G}$ is associated with non-negative demand Q_g, and let $\mathcal{D}$ be a family of n sets where each set $t \in \mathcal{D}$ is defined as a subset of $\mathcal{G}$, i.e., $t \subseteq \mathcal{G}$, and it is associated with a non-negative weight w_t. The goal is to select a family of sets $\mathcal{H} \subseteq \mathcal{D}$ (each set in $\mathcal{D}$ may be selected at most once in $\mathcal{H}$) with minimum total weight such that every item $g \in \mathcal{G}$ is covered at least Q_g times. Formally, that is:*

$$\textit{minimize:} \quad \sum_{t\in\mathcal{H}} w_t \tag{1}$$

$$\textit{subject to:} \quad |\{t \in \mathcal{H} \mid g \in t\}| \geq Q_g, \quad \forall g \in \mathcal{G} \tag{2}$$

$$\mathcal{H} \subseteq \mathcal{D}. \tag{3}$$

Throughout the paper, for simplicity, we always assume $Q_g \leq n$, for all $g \in \mathcal{G}$, since otherwise, the instance becomes infeasible. Note that $\mathcal{D}$ may contain multiple distinct elements that represent

[2]Using the fastest currently known algorithm [30], we have $\mathcal{L}(n) = O(n^{2.38})$.

the same underlying set, possibly with identical or different weights. Each such element is treated as a separate copy. In the solution $\mathcal{H}$, at most one instance of each copy may be selected; that is, no copy can be used more than once, but different copies containing the same set of items can be used simultaneously. For a positive integer z, we define $[z] = \{1, \ldots, z\}$. We summarize all commonly used notation in Table 1.

An optimum solution to the WSMC-BU problem is a set of tuples $\mathcal{H}^*$ that minimizes the objective function (1) satisfying the constraints (2), (3).

An algorithm for the WSMC-BU problem is called an α-approximation algorithm (for any $\alpha \geq 1$), if it returns a set $\mathcal{H}'$ that satisfies the constraints (2), (3) and $\sum_{t' \in \mathcal{H}'} w_{t'} \leq \alpha \cdot \sum_{t^* \in \mathcal{H}^*} w_{t^*}$. The set $\mathcal{H}'$ is also called an α-approximation solution for the WSMC-BU problem.

Notation	Definition
$\mathcal{G}$	Universe of items
ℓ	Size of the universe which is constant
Q_g	Demand of item $g \in \mathcal{G}$
$\mathcal{D}$	Family of sets (subsets of $\mathcal{G}$)
n	Number of sets in $\mathcal{D}$
w_t	Weight of set $t \in \mathcal{D}$
W	$\frac{\sum_{t \in \mathcal{D}} w_t}{\min_{t \in \mathcal{D}} w_t}$
$\mathcal{L}(X)$	LP solve time with X variables and $O(X)$ constraints
$[X]$	$\{1, \ldots, X\}$
$B[H]$	All sets in $\mathcal{D}$ that contain the subset of items $H \subseteq \mathcal{G}$
ε	Arbitrary positive constant

Table 1. Table of Notations

2.2 Overview

In the next sections, we present a simple exact algorithm and two more involved approximation algorithms for the WSMC-BU problem. We briefly show the high-level ideas of our methods

For the exact algorithm, we discuss the natural dynamic programming algorithm for the WSMC-BU problem.

Next, we study approximation algorithms. We begin by formulating WSMC-BU as an Integer Program and relaxing it to an optimization problem that minimizes the sum of non-decreasing, convex, piecewise linear functions subject to linear constraints. We show that this relaxation can be solved exactly by formulating it as a Linear Program (LP). We solve this LP (using an LP-solver) and design a novel rounding technique that yields a 2-approximation algorithm for the WSMC-BU problem. The running time of our algorithm is $O(\mathcal{L}(n))$, where $\mathcal{L}(n)$ is the time to solve an LP with n variables and $O(n)$ constraints.

The LP we obtain is related to the standard LP (15) for the weighted set multi-cover problem, although it has a different formulation. Furthermore, while randomized rounding on the standard LP (15) yields an $O(\log \ell)$-approximation, our novel rounding technique can also be applied to the standard formulation and improves the guarantee to 2. The motivation for formulating our problem as the minimization of a sum of piecewise linear functions under linear constraints will become clear in the following paragraph.

Our algorithm is the first truly constant-factor approximation (independent of ℓ) for the WSMC-BU problem. However, the running time is super-quadratic in n. To address this, we further refine our approach. Instead of directly solving the relaxed optimization problem with the original piecewise linear functions, we approximate these functions within a $(1 + \varepsilon)$ factor by constructing new

non-decreasing, convex, piecewise linear functions with fewer linear pieces. The motivation for introducing this formulation, rather than working directly with the standard LP, is that it allows us to describe the piecewise functions more compactly, leading to smaller LPs and thus faster running time. The resulting LP is solved optimally, and rounding its solution yields a $(2+\varepsilon)$-approximation algorithm for the WSMC-BU problem with running time $O(n \log n + \mathcal{L}(\log W))$, where $W = \frac{\sum_{t \in \mathcal{D}} w_t}{\min_{t \in \mathcal{D}} w_t}$.

3 Exact Algorithms

As previously mentioned, [4] showed that the Weighted Set Multi-Cover problem is fixed parameter tractable when parameterized by the size of the universe of elements to cover. This implies that when the size of the universe is a constant, there exists an algorithm for solving the Weighted Set Multi-Cover problem in polynomial time with respect to the size of the input, or equivalently, there exists a polynomial time algorithm for solving WSMC-BU problem. Although theoretically intriguing, their approach lacks practical applicability, as the exact polynomials are not analyzed, and both the constant factors and polynomial exponents appear to be significantly large.

In this section, as a warm-up, we present a simple exact algorithm to our problem using Dynamic Programming (DP). While the DP approach is not always faster than the optimum algorithm in [4], it is significantly simpler and easier to implement. In contrast, we are not aware of an implementation of the optimum algorithm in [4]. The DP algorithm runs in time exponential to $|\mathcal{G}|$. Since $|\mathcal{G}|$ is constant, the running time is polynomial on n.

We aim to develop an optimal DP-based strategy that minimizes the total weight of the WSMC-BU problem. This process involves a sequence of iterative steps in which the algorithm evaluates the contribution of each set by updating the residual demands and the DP value at each step. In each iteration, the optimal solution records the minimum weight needed to fulfill the demands for the given state, up to the i-th set (arbitrary order). Each state is defined as the current level of demands satisfied for each item in $\mathcal{G}$.

For simplicity, we present the algorithm to compute the weight of the optimum solution. It is trivial how to restore the actual optimum family of subsets that achieve the optimum weight by storing the history of updates. Let $t_1, \ldots, t_n$ be an arbitrary ordering of the sets in $\mathcal{D}$. We define the table DP with dimensions $(n+1) \times (n+1) \times \ldots \times (n+1)$ ($\ell+1$ times). Let $\mathrm{DP}[i][q_1]\ldots[q_\ell]$ be the weight of the optimum solution of the WSMC-BU problem considering the first i sets with demands $q_1, \ldots, q_\ell$ on items $g_1, \ldots, g_\ell \in \mathcal{G}$, respectively. We first initialize all cells in table DP to ∞. By definition, we set $\mathrm{DP}[i][0]\ldots[0] = 0$ for every $i \in \{0\} \cup [n]$. For every $i \in [n]$, $q_1 \in \{0\} \cup [n], \ldots, q_\ell \in \{0\} \cup [n]$, it holds,

$$\begin{aligned}\mathrm{DP}[i][q_1]\ldots[q_\ell] = \min\{&\mathrm{DP}[i-1][q_1]\ldots[q_\ell], w_{t_i} + \\ &\mathrm{DP}[i-1][\max\{0, q_1 - \mathbb{1}[g_1 \in t_i]\}]\ldots[\max\{0, q_\ell - \mathbb{1}[g_\ell \in t_i]\}]\},\end{aligned} \tag{4}$$

where $\mathbb{1}[\text{condition}]$ is the indicator function that returns 1 if the condition holds, and 0 otherwise. In other words, in every step, we decide whether we include the set t_i in the current solution. The optimum weight among the first i sets and demands $q_1, \ldots, q_\ell$ is the minimum between i) Do not take set t_i: the optimum weight among the first $i-1$ sets and demands $q_1, \ldots, q_\ell$, and ii) Take set t_i: weight of set t_i plus the optimum weight among the first $i-1$ sets and demands $q_1 - \mathbb{1}[g_1 \in t_i], \ldots, q_\ell - \mathbb{1}[g_\ell \in t_i]$. After filling all cells in table DP, we compute the optimum weight as $\min_{j_1 \geq Q_{g_1}, \ldots, j_\ell \geq Q_{g_\ell}} \mathrm{DP}[n][j_1]\ldots[j_\ell]$. By tracing the optimum path in the DP algorithm, we get the family $\mathcal{H}$ of the optimum selected sets. The pseudocode of the algorithm is presented in the technical report [51].

The correctness of the algorithm follows from Equation (4). For the running time, we note that DP has $O(n^{|\mathcal{G}|+1})$ cells. In each cell we spend $O(\ell) = O(1)$ time to set the weight according to Equation 4. Overall, the running time of the algorithm is $O(n^{|\mathcal{G}|+1})$. We conclude with the following theorem.

Theorem 1. *There exists an exact algorithm for the WSMC-BU problem that runs in $O(n^{|\mathcal{G}|+1})$ time.*

4 Approximation Algorithms

While the straightforward DP algorithm from the previous section runs in polynomial time, it quickly becomes inefficient when $|\mathcal{G}|$ is a large constant. Consequently, we turn our attention to developing efficient approximation algorithms and demonstrate their theoretical guarantees and practicality. We first show a 2-approximation algorithm that runs in $O(\mathcal{L}(n))$ time, where $\mathcal{L}(x)$ is the running time to solve a Linear Program with $O(x)$ variables and $O(x)$ constraints. Then, we show a $(2+\varepsilon)$-approximation algorithm that runs in $O(n \cdot \log n + \mathcal{L}(\log(W))$ time, where ε is an arbitrarily small constant given by the user, say 0.01, and $W = \frac{\sum_{t\in\mathcal{D}} w_t}{\min_{t\in\mathcal{D}} w_t}$.

4.1 Polynomial 2-approximation algorithm

In this section, we show an LP-based approximation algorithm for the WSMC-BU problem. We first model WSMC-BU as an Integer Program (IP), and then relax the constraints to allow fractional solutions. We then show that the relaxed program can be written as a Linear Program (LP). We use an LP-solver to compute the optimum solution of the LP, and we use rounding to get a 2-approximation solution. While we could directly formulate the WSMC-BU problem as an Integer Linear Program (ILP) and then solve its LP relaxation, we use a different formulation that will allow us to improve the running time of the algorithm in the next sections. We note that the main goal of the algorithm discussed in this subsection is to provide a foundation for the efficient algorithm discussed in Subsection 4.2.

We first partition the sets from $\mathcal{D}$ in buckets based on the items they contain. For every subset $H \subseteq \mathcal{G}$, let $B(H) = \{t \in \mathcal{D} \mid t = H\}$, i.e., the bucket $B(H)$ contains all sets in $\mathcal{D}$ that contain exactly the subset of items H. Let $\mathcal{B} = \{B(H) \mid H \subseteq \mathcal{G}\}$. Since $|\mathcal{G}| = \ell$, there are at most 2^ℓ different buckets, $|\mathcal{B}| = O(2^\ell) = O(1)$. For every subset $H \subseteq \mathcal{G}$, we sort the sets in $B(H)$ by their weights in ascending order and we define the function $f_{B(H)}(x)$ that returns the sum of the weights of the first x sets in $B(H)$, for every $x \in \{0, 1, \ldots, |B(H)|\}$. Note that $f_{B(H)}(0) = 0$, for all $H \subseteq \mathcal{G}$. For simplicity, we refer to $f_{B(H)}(\cdot)$ as $f_H(\cdot)$. In other words, $f_H(x)$ returns the cost of choosing x sets from $B(H)$ with the minimum weight.

With these definitions, we can now model the WSMC-BU as an IP. For each $H \subseteq \mathcal{G}$, let x_H be the integer variable that denotes the number of sets selected from $B(H)$ in the returned solution. The WSMC-BU problem is equivalent to the following Integer Program.

$$\text{minimize:} \quad \sum_{H\subseteq\mathcal{G}} f_H(x_H) \tag{5}$$

$$\text{subject to:} \quad \sum_{H\subseteq\mathcal{G},\, g\in H} x_H \geq Q_g, \quad \forall g \in \mathcal{G} \tag{6}$$

$$0 \leq x_H \leq |B(H)|, \quad \forall H \subseteq \mathcal{G} \tag{7}$$

$$x_H \in \mathbb{Z}, \quad \forall H \subseteq \mathcal{G}. \tag{8}$$

Constraints (6) ensure that the demands of all items in $\mathcal{G}$ are satisfied. Constraints (7) ensure that we select at most $|B(H)|$ sets from each subset $H \subseteq \mathcal{G}$. If we select $\sigma_H \in \mathbb{Z}$ sets from the bucket

Expert #	Skills	Hourly Rate
1	(Legal, Medical)	5$
2	(Legal, Medical)	13$
3	(Legal, Medical)	17$
4	(Legal, Medical)	20$
5	(Legal, Medical)	25$
6	(Legal, Medical)	50$

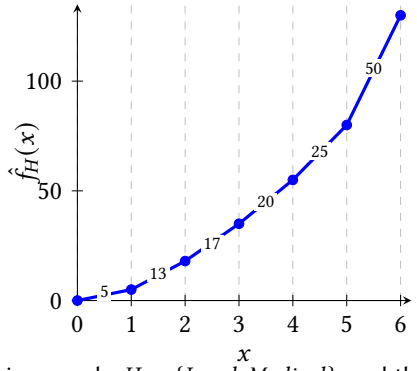


Fig. 2. Sample of experts with the same skill set. In this example, $H = \{Legal, Medical\}$, and the rate is considered as the weight. The figure shows the piecewise linear function $\hat{f}_H(x)$, based on the shown table.

$B(H)$, it is always optimal to use the sets with minimum weight possible, and since $f_H(\sigma_H)$ returns the total weight of the lightest σ_H sets from $B(H)$, the objective function in (5) returns the total weight of the solution. Therefore, the problem formulation (5) correctly captures the WSMC-BU problem.

Next, we relax the IP (5) to allow fractional solutions, and then show a rounding scheme to round the fractional solution into an integral solution while preserving the accuracy. In the next paragraphs, let opt be the value of the objective function in the optimum solution of IP (5).

Relaxation. We note that every function $f_H : \mathbb{Z} \to \mathbb{R}$ is only defined on integral values $\{0, 1, \ldots, B(H)\}$. For each $H \subseteq \mathcal{G}$, we define a real function $\hat{f}_H : \mathbb{R} \to \mathbb{R}$ to evaluate fractional solutions:

$$\hat{f}_H(x) = (f_H(z+1) - f_H(z)) \cdot (x - z) + f_H(z), \quad \text{if } z \le x \le z+1, z \in \mathbb{Z}.$$

Using the function $\hat{f}_H(\cdot)$, we define the following optimization problem.

$$\begin{aligned} \text{minimize:} \quad & \sum_{H \subseteq \mathcal{G}} \hat{f}_H(x_H) \\ \text{subject to:} \quad & \sum_{H \subseteq \mathcal{G}, g \in H} x_H \ge Q_g, \quad \forall g \in \mathcal{G}, \\ & 0 \le x_H \le |B(H)|, \quad \forall H \subseteq \mathcal{G} \\ & x_H \in \mathbb{R}, \quad \forall H \subseteq \mathcal{G}. \end{aligned} \tag{9}$$

Let $\hat{\text{opt}}$ be the value of the objective function in the optimum solution of Problem (9).

An example of $\hat{f}_H(\cdot)$ is shown in Figure 2. As we show later, for every $H \subseteq \mathcal{G}$, function $\hat{f}_H(\cdot)$ is non-decreasing, convex and piecewise linear. Hence, we argue that Problem (9) is an abbreviated representation of a pure LP. There are ways to translate Problem 9 into a pure Linear Program (see for example the λ method from [34, 61]). More specifically, for each $H \subseteq \mathcal{G}$, we define the binary variable $x_{H,i}$ which is 1 if the set with the i-th smallest weight in $B(H)$ is selected in the solution, and 0 otherwise. Let $w_{H,i}$ be the weight of the i-th smallest weight in $B(H)$. We define the following LP.

$$\text{minimize:} \quad \sum_{H \subseteq \mathcal{G}} \sum_{i \in [|B(H)|]} w_{H,i} \cdot x_{H,i} \tag{10}$$

$$\text{subject to:} \quad \sum_{H \subseteq \mathcal{G}, g \in H} \sum_{i \in [|B(H)|]} x_{H,i} \geq Q_g, \quad \forall g \in \mathcal{G}, \tag{11}$$

$$0 \leq \sum_{i \in [|B(H)|]} x_{H,i} \leq |B(H)|, \quad \forall H \subseteq \mathcal{G} \tag{12}$$

$$0 \leq x_{H,i} \leq 1, \quad \forall H \subseteq \mathcal{G}, i \in [|B(H)|]. \tag{13}$$

We note that for every $H \subseteq \mathcal{G}$, there are $|B(H)|$ variables, and every set in $\mathcal{D}$ belongs to exactly one bucket $B(H)$. Hence the total number of variables are n. Since $|\mathcal{G}| = \ell = O(1)$ there are $O(1)$ non-trivial type (11) constraints. Furthermore, there are $O(2^\ell) = O(1)$ subsets of $\mathcal{G}$ so there are $O(1)$ non-trivial type (12) constraints. Finally, there are $O(n)$ trivial type (13) constraints.

From now on, given a set of real numbers $\{y_H \mid H \subseteq \mathcal{G}\}$, we use the notation $\vec{y}_{H \subseteq \mathcal{G}} \in \mathbb{R}^{2^\ell}$ to denote the vector whose coordinate corresponding to a subset $H \subseteq \mathcal{G}$ takes the value y_H.

The proof of the next lemma is shown in the technical report [51].

Lemma 1. *(i)* $\hat{\text{opt}} \leq \text{opt}$. *(ii) For every* $H \subseteq \mathcal{G}$, *the function* $\hat{f}_H(\cdot)$ *is a non-decreasing piecewise linear and convex function. (iii) Problem* (9) *is equivalent to LP* (10).

Using the observations from Lemma 1, we describe the algorithm as follows. We first form the LP (10). Using an LP solver, we get the optimum solution. For each $H \subseteq \mathcal{G}$ and $i \in [|B(H)|]$, let $\hat{x}_{H,i}$ be the value of variable $x_{H,i}$ in the optimum solution. From the proof of Lemma 1, LP (10) and Problem (9) are equivalent, so without loss of generality, assume that for each $H \subseteq \mathcal{G}$, $\hat{x}_H$ is the value of variable x_H in the optimum solution of Problem (9).

Running Example 1: Assume an instance of the WSMC-BU problem with $\mathcal{G} = \{g_1, g_2\}$. For $H_1 = \{g_1\} \in \mathcal{G}$ there exists three sets $\Gamma_1 = \Gamma_2 = \Gamma_3 = \{g_1\}$ in $\mathcal{D}$ with weights $1, 3, 4$, respectively. The demand of g_1 is $Q_{g_1} = 3$. For $H_2 = \{g_2\} \in \mathcal{G}$ there exists two sets $\Delta_1 = \Delta_2 = \{g_2\}$ in $\mathcal{D}$ with weights $2, 8$ respectively. The demand of g_2 is $Q_{g_2} = 2$. Finally, for $H_{1,2} = \{g_1, g_2\} \in \mathcal{G}$ there exist three sets $E_1 = E_2 = E_3 = \{g_1, g_2\}$ in $\mathcal{D}$ with weights $4, 5, 6$, respectively.

We first present the formulation of Problem 9 under the setting described above. In the toy example, Problem 9 is defined over 3 variables, i.e., one for every subset of $\mathcal{G}$. Namely, we have the variables x_{H_1}, x_{H_2}, and $x_{H_{1,2}}$.

In order to compute the objective functions we should define $\hat{f}_H(x)$ for every $H \subseteq \mathcal{G}$. For $H_1 = \{g_1\} \subset \mathcal{G}$, $f_{H_1}(0) = 0$, $f_{H_1}(1) = 1$, $f_{H_1}(2) = 4$, and $f_{H_1}(3) = 8$, so the function $\hat{f}_{H_1}(x)$ is defined as follows: i) If $0 \leq x < 1$, then $\hat{f}_{H_1}(x) = x$, ii) if $1 \leq x < 2$, then $\hat{f}_{H_1}(x) = 3(x-1) + 1 = 3x - 2$, and iii) if $2 \leq x \leq 3$, then $\hat{f}_{H_1}(x) = 4(x-2) + 4 = 4x - 4$. For $H_2 = \{g_2\} \subset \mathcal{G}$, then $f_{H_2}(0) = 0$, $f_{H_2}(1) = 2$, and $f_{H_2}(2) = 10$, so the function $\hat{f}_{H_2}(x)$ is defined as follows: i) If $0 \leq x < 1$, then $\hat{f}_{H_2}(x) = 2x$, and ii) if $1 \leq x \leq 2$, then $\hat{f}_{H_2}(x) = 8(x-1) + 2 = 8x - 6$. For $H_{1,2} = \{g_1, g_2\} = \mathcal{G}$, then $f_{H_{1,2}}(0) = 0$, $f_{H_{1,2}}(1) = 4$, $f_{H_{1,2}}(2) = 9$, and $f_{H_{1,2}}(3) = 15$, so the function $\hat{f}_{H_{1,2}}(x)$ is defined as follows: i) If $0 \leq x < 1$, then $\hat{f}_{H_{1,2}}(x) = 4x$, ii) if $1 \leq x < 2$, then $\hat{f}_{H_{1,2}}(x) = 5(x-1) + 4 = 5x - 1$, and iii) if $2 \leq x \leq 3$, then $\hat{f}_{H_{1,2}}(x) = 6(x-2) + 9 = 6x - 3$.

Next, we show the constraints of Problem 9. For $g_1 \in \mathcal{G}$, we must satisfy $x_{H_1} + x_{H_{1,2}} \geq 3$, while for $g_2 \in \mathcal{G}$ we must satisfy $x_{H_2} + x_{H_{1,2}} \geq 2$. Finally, we must have $0 \leq x_{H_1} \leq 3$, $0 \leq x_{H_2} \leq 2$, and $0 \leq x_{H_{1,2}} \leq 3$.

The first phase of our algorithm computes the optimum solution to this optimization problem using an LP-solver.

Rounding scheme. Next, we present the rounding scheme of our algorithm. While our rounding algorithm is theoretically efficient, as shown in Section 5, in practice the LP formulated in the previous paragraphs almost always produces small fractional residuals, so our rounding scheme remains efficient in practical settings.

Let $\mathcal{H}$ be an initially empty set denoting the family of sets we choose in our solution. We define $\bar{x}_H = \lfloor \hat{x}_H \rfloor$ for all $H \subseteq \mathcal{G}$, and from $B(H)$, we add the $\bar{x}_H$ sets with the smallest weight to the solution set $\mathcal{H}$. By rounding down the values $\hat{x}_H$, some demands may remain unsatisfied. Next, we describe how we add more sets to the set $\mathcal{H}$ to make sure all the demands are satisfied.

Let $\bar{B}(H) = B(H) - \mathcal{H}$, be the family of unused sets in each bucket for all $H \subseteq \mathcal{G}$, and let $r = \left\lceil \sum_{H \subseteq \mathcal{G}} (\hat{x}_H - \bar{x}_H) \right\rceil$. From each bucket $\bar{B}(H)$, we add at most r more sets to the solution set. To find the sets we add to the solution, we brute force all the possible ways of choosing between 0 to r sets from each bucket $\bar{B}_H$, and among the possible family of sets that satisfy all the demands, we choose the sets having the minimum total weight and add them to the solution set $\mathcal{H}$. We will later show why it is sufficient to consider at most r additional sets from each bucket to satisfy all the remaining demands. The pseudo-code of our algorithm is shown in Algorithm 1.

In the technical report [51], we prove the following theorem.

Theorem 2. *Given an instance $(\mathcal{D}, \mathcal{G})$ of the WSMC-BU problem with $|\mathcal{D}| = n$ and $|\mathcal{G}| = O(1)$, there exists a 2-approximation algorithm for the WSMC-BU problem that runs in $O(\mathcal{L}(n))$ time.*

Running Example 1 (cont.) Next, we show an execution of our rounding scheme using the toy example we defined in Running Example 1. For simplicity, assume that the optimum solution found by solving Problem 9 is $\hat{x}_{H_1} = 1.5$, $\hat{x}_{H_2} = 0.5$, and $\hat{x}_{H_{1,2}} = 1.5$. We note that this is not the optimum solution, however we will use this feasible solution to demonstrate our rounding scheme. We have $\bar{x}_{H_1} = 1$, $\bar{x}_{H_2} = 1$, and $\bar{x}_{H_{1,2}} = 0$. In the set $\mathcal{H}$ we add the set H_1 with the smallest weight, which is Γ_1 and the set $H_{1,2}$ with the smallest weight, which is E_1. Hence $\mathcal{H} = \{\Gamma_1, E_1\}$. We also have $\bar{B}(H_1) = \{\Gamma_2, \Gamma_3\}$, $\bar{B}(H_2) = \{\Delta_1, \Delta_2\}$, and $\bar{B}(H_{1,2}) = \{E_2, E_3\}$. By the definition of r we get $r = \lceil 1.5 \rceil = 2$. Next, we brute force all combinations choosing between 0 and $r = 2$ sets from each bucket $\bar{B}(H_1)$, $\bar{B}(H_2)$, $\bar{B}(H_{1,2})$, and among the family of sets that satisfy the demands we choose the set with the minimum total weight. For example, the family that contains the pre-selected sets in $\mathcal{H}$ along with 2 sets from $\bar{B}(H_1)$ (sets Γ_2, Γ_3), and no set from $\bar{B}(H_{1,2})$ and $\bar{B}(H_2)$ does not satisfy the demands. Notice that while the demand of g_2 is $Q_{g_2} = 2$ there is only one set in $\mathcal{H} \cup \{\Gamma_2\} \cup \{\Gamma_3\} = \{\Gamma_1, \Gamma_2, \Gamma_3, E_1\}$ that contains g_2. On the other hand, the family that contains the pre-selected sets in $\mathcal{H}$ along with 1 set from $\bar{B}(H_1)$ (set Γ_2), 1 set from $\bar{B}(H_{1,2})$ (set E_2), and no set from $\bar{B}(H_2)$ satisfy the demands. Notice that in $\mathcal{H} \cup \{\Gamma_2\} \cup \{E_2\} = \{\Gamma_1, \Gamma_2, E_1, E_2\}$ there exist $4 > Q_{g_1}$ sets that contain g_1 and $2 \geq Q_{g_2}$ sets that contain g_2. However, the total weight of the selected family of sets $\{\Gamma_1, \Gamma_2, E_1, E_2\}$ is 13, which is not the minimum. By trying all combinations, we find the the optimum family is the one that contains the pre-selected sets in $\mathcal{H}$ along with 1 set from $\bar{B}(H_1)$ (set Γ_2), 1 set from $\bar{B}(H_2)$ (set Δ_1), and no set from $\bar{B}(H_{1,2})$, i.e, the family $\mathcal{H} \cup \{\Gamma_2\} \cup \{\Delta_1\} = \{\Gamma_1, \Gamma_2, E_1, \Delta_1\}$ satisfies all the demands and its total weight is 10.

4.2 Fast $(2+\epsilon)$-approximation

In this section, we show how we can use a slightly different LP formulation to achieve a much faster algorithm. As discussed, the running time bottleneck of Algorithm 1 is solving the LP (10). The main complexity of the LP formulation comes from the number of variables, equivalently, the number of linear pieces in functions $\hat{f}_H(\cdot)$ from Problem (9). As discussed in Subsection 4.1, the total number of different linear pieces across all $H \subseteq \mathcal{G}$ is $O(n)$. This makes the size of LP (10) to be $O(n)$, and solving this program takes super-quadratic time in terms of the number of sets n. The idea of the algorithm we discuss in this section is to describe each function $\hat{f}_H(\cdot)$ with a smaller

Algorithm 1 LP-based 2-approximation Algorithm

Input: $\mathcal{D}, \mathcal{G}, \{Q_g \mid g \in \mathcal{G}\}$
Output: Family of selected sets $\mathcal{H}$

Form and solve LP (10)
Convert the solution from LP (10) to a solution for Problem 9
For each $H \subseteq \mathcal{G}$, let $\bar{x}_H$ be the value of the variable x_H in the optimum solution of Problem 9
$\mathcal{H} \leftarrow \emptyset$
for all $H \subseteq \mathcal{G}$ **do**
    $\bar{x}_H \leftarrow \lfloor \hat{x}_H \rfloor$
    Add $\bar{x}_H$ sets from $B(H)$ with minimum weight to $\mathcal{H}$
for all $H \subseteq \mathcal{G}$ **do**
    $\bar{B}(H) \leftarrow B(H) \setminus \mathcal{H}$
$r \leftarrow \left\lceil \sum_{H \subseteq \mathcal{G}} (\hat{x}_H - \bar{x}_H) \right\rceil$
$W' \leftarrow \infty, \mathcal{H}' \leftarrow \emptyset$
for all integer vectors $\vec{y}_{H \subseteq \mathcal{G}}$ such that
$0 \leq y_H \leq \min\{r, |\bar{B}(H)|\}$ **do**
    $S \leftarrow \emptyset$
    for all $H \subseteq \mathcal{G}$ **do**
        Add y_H sets with minimum weight from $\bar{B}(H)$ to S
    for all $g \in \mathcal{G}$ **do**
        $q_g \leftarrow \sum_{H \ni g} \bar{x}_H + y_H$
    if $q_g \geq Q_g$ for all $g \in \mathcal{G}$ **then**
        $w \leftarrow$ total weight of S
        if $w < W'$ **then**
            $W' \leftarrow w, \mathcal{H}' \leftarrow S$
$\mathcal{H} \leftarrow \mathcal{H} \cup \mathcal{H}'$
return $\mathcal{H}$

number of linear pieces bounding the error. This will allow us to significantly reduce the size of the LP instance and solve it efficiently.

Recall that every function $\hat{f}_H$ is convex, non-decreasing, and piecewise linear. We start by describing a more general result of approximating any continuous, non-decreasing, convex function (not necessarily piecewise linear) with a continuous, non-decreasing, convex, and piecewise linear function. Then, we use this result to approximate the piecewise linear function $\hat{f}_H(\cdot)$ with another continuous, non-decreasing, convex, and piecewise linear function having less number of linear pieces.

Approximating functions by piecewise linear functions. Let $f : [\alpha, \beta] \rightarrow [\gamma, \delta]$ be any continuous, convex, and non-decreasing function such that $0 < \alpha \leq \beta$ and $0 < \gamma \leq \delta$, and let $\varepsilon > 0$ be any constant. We describe an algorithm to approximate the function $f(\cdot)$ by a piecewise linear function $g(\cdot)$, such that $f(x) \leq g(x) \leq (1+\epsilon) f(x)$, for any value of $\alpha \leq x \leq \beta$.

We start by $x_1 = \alpha$ and get $f(x_1)$. Then, we find the largest value x'_1, such that $f(x'_1) \leq (1+\epsilon) f(x_1)$ and $x'_1 \leq \beta$. Since the function $f(\cdot)$ is non-decreasing and continuous it is sufficient to only consider $x'_1 > x_1$. Let $\text{LargestVal}_f(x_1, (1+\varepsilon) f(x_1))$ be a procedure over the function f that returns such value x'_1. Next, we set $x_2 = x'_1$, and recursively repeat the same process using x_2 instead of x_1, calling $\text{LargestVal}_f(x_2, (1+\varepsilon) f(x_2))$. More generally, at the i'th step, we find the largest

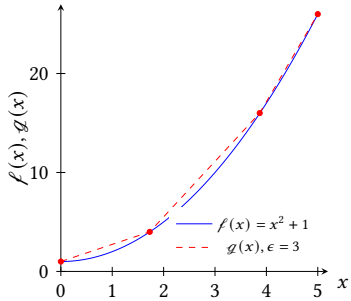


Fig. 3. Approximation of the $f(x) = x^2 + 1$ curve by a piecewise linear function $g(\cdot)$, generated by the described algorithm, where $\epsilon = 3$. The function $g(\cdot)$ approximates the values of the function $f(\cdot)$ within a factor of 4.

value $x_i' \leq \beta$, such that $f(x_i') \leq (1+\epsilon) f(x_i)$, and then set $x_{i+1} = x_i'$. We continue this process until $x_s = \beta$ for some step s. Finally, we set the function $g(\cdot)$ to contain $s-1$ linear pieces where the i-th piece of $g(\cdot)$ is the line segment connecting the points $(x_i, f(x_i))$ and $(x_{i+1}, f(x_{i+1}))$, for every $i \in [s-1]$. An illustration is shown in Figure 3. We prove the next lemma in the technical report [51].

Lemma 2. *Given a parameter $\varepsilon > 0$, any continuous, convex and non-decreasing function $f(\cdot) : [\alpha, \beta] \to [\gamma, \delta]$, where $0 < \alpha \leq \beta$ and $0 < \gamma \leq \delta$, can be approximated by a continuous, convex non-decreasing piecewise linear function $g(\cdot) : [\alpha, \beta] \to [\gamma, \delta]$ having $O(\log(\frac{\delta}{\gamma}) \log^{-1}(1+\varepsilon))$ pieces, such that for every $\alpha \leq x \leq \beta$, it holds that $f(x) \leq g(x) \leq (1+\epsilon) f(x)$. Moreover, the function $g(\cdot)$ can be constructed in $O(\log(\frac{\delta}{\gamma}) \cdot \log^{-1}(1+\varepsilon) \cdot T_f^{>} \cdot T_f^{=})$ time, where $T_f^{=}$ is the time needed to calculate $f(x)$, for any $\alpha \leq x \leq \beta$, and $T_f^{>}$ is the time needed to compute the smallest value $x' \geq \bar{x}$ such that $f(x') \leq (1+\varepsilon) f(\bar{x})$, for any values $\bar{x}, x'$ satisfying $\alpha \leq \bar{x} \leq x' \leq \beta$.*

Our algorithm. We move back to our original problem. For all $H \subseteq \mathcal{G}$, we keep the first linear piece, while for the rest pieces we use Lemma 2 to approximate $\hat{f}_H(\cdot)$. Let $\hat{g}_H(\cdot)$ be the resulting piecewise linear function we obtain from Lemma 2 along with the first linear piece from $\hat{f}_H(\cdot)$. By Lemma 2, for all $x \in [0, |B(H)|]$, it holds that $\hat{f}_H(x) \leq \hat{g}_H(x) \leq (1+\epsilon) \hat{f}_H(x)$, where $\varepsilon > 0$ is a constant chosen by the user. We then define the optimization problem, similarly to Problem 9 as shown in Subsection 4.1, but we use the functions $\hat{g}_H(\cdot)$ instead of $\hat{f}_H(\cdot)$.

$$\begin{aligned} \text{minimize:} \quad & \sum_{H \subseteq \mathcal{G}} \hat{g}_H(x_H) \\ \text{subject to:} \quad & \sum_{H \subseteq \mathcal{G},\, g \in H} x_H \geq Q_g, \quad \forall g \in \mathcal{G}, \\ & 0 \leq x_H \leq |B(H)|, \quad \forall H \subseteq \mathcal{G} \\ & x_H \in \mathbb{R}. \end{aligned} \tag{14}$$

Using our machinery from Subsection 4.1 along with the proof of Lemma 1, we can formulate Problem (14) as an LP. We then solve the new LP to get the fractional solution. Finally, we round its solution using the same rounding scheme as in Algorithm 1.

Running Example 1 (cont.) We continue Example 1 executing our new algorithm. For every subset $H = \{H_1, H_2, H_{1,2}\}$, we construct the function $\hat{g}_H(x)$. For simplicity we focus on $H_1 = \{g_1\}$, however the same approach works for the rest subsets. Recall the definition of $\hat{f}_{H_1}(x)$ which is a piecewise linear function with three pieces as shown in Figure 4. For $x \in [0, 1]$ by the description

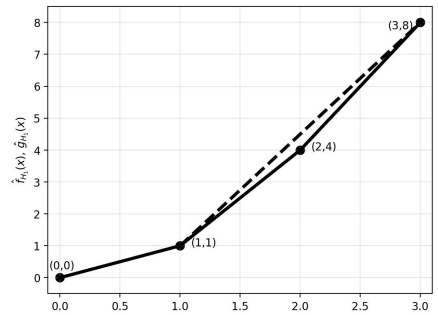


Fig. 4. The function $\hat{f}_{H_1}(x)$ is a piecewise linear function consisting of three solid segments. For $\varepsilon = 7$, the function $\hat{g}_{H_1}(x)$ is a piecewise linear function with two segments: for $x \in [0, 1]$, $\hat{g}_{H_1}(x) = \hat{f}_{H_1}(x)$, while for $x \in [1, 3]$, $\hat{g}_{H_1}(x)$ is given by the dashed linear segment connecting $(1, 1)$ and $(3, 8)$.

of our algorithm we have $\hat{g}_{H_1}(x) = \hat{\hat{f}}_{H_1}(x) = x$. Then for the rest of the pieces we execute the algorithm from Lemma 2. Assume that $\varepsilon = 7$. We have $\alpha = 1, \beta = 3, \gamma = 1$, and $\delta = 8$. Starting from $x_1 = 1$ we compute the largest $x_1' \leq 3$ such that $\hat{f}_{H_1}(x_1') \leq (1 + \varepsilon)\hat{\hat{f}}_{H_1}(x_1) = 8 \cdot \hat{\hat{f}}_{H_1}(x_1)$ and $x_1' \leq 3$. We observe that $x_1' = 3$ so $\hat{g}_{H_1}(x)$ for $x \in [1, 8]$ is defined as the linear segment that connects $(1, 1)$ and $(3, 8)$, as shown with the dashed segment in Figure 4. Repeating the same procedure for every subset H of $\mathcal{G}$ we define all functions $\hat{g}_H(x)$. The constraints of Problem 14 are the same with the constraints of Problem 9.

Analysis. For every $H \subseteq \mathcal{G}$, let $\tilde{x}_H$ be the integer value of the variable x_H that is returned by the rounding scheme. Recall that opt is the value of the optimum solution in IP (5). The next lemmas are shown in the technical report [51].

Lemma 3. $\sum_{H \subseteq \mathcal{G}} f_H(\tilde{x}_H) \leq 2 \cdot (1 + \varepsilon) \cdot \text{opt}$.

Notice that if we run the algorithm from Lemma 2, setting $\varepsilon \leftarrow \varepsilon/2$, we will get directly $\sum_{H \subseteq \mathcal{G}} f_H(\tilde{x}_H) \leq (2 + \varepsilon)\text{opt}$.

Lemma 4. *For any arbitrary constant $\varepsilon > 0$, the algorithm runs in $O(n \log n + \mathcal{L}(\log(W)))$ time, where $W = \frac{\sum_{t \in \mathcal{D}} w_t}{\min_{t \in \mathcal{D}} w_t}$.*

Putting everything together, we conclude to the next theorem.

Theorem 3. *Given an instance $(\mathcal{D}, \mathcal{G})$ of the WSMC-BU problem with $|\mathcal{D}| = n$ and $|\mathcal{G}| = O(1)$, and an arbitrary constant $\varepsilon > 0$, there exists a $(2 + \varepsilon)$-approximation algorithm for the WSMC-BU problem that runs in $O(n \log n + \mathcal{L}(\log(W)))$ time, where $W = \frac{\sum_{t \in \mathcal{D}} w_t}{\min_{t \in \mathcal{D}} w_t}$.*

5 Evaluation

5.1 Experiment Plan

We aim to evaluate our algorithms with respect to both effectiveness and efficiency. The effectiveness experiments assess the quality of the solutions obtained, specifically in terms of total solution weight, while ensuring that the required demands are satisfied. We further assess the extent to which the obtained solution deviates from the specified demand requirements. The efficiency experiments, on the other hand, measure the computational performance of each algorithm, i.e., the time required to obtain a solution. For each experiment, we evaluate the impact of varying the following parameters: number of sets in the dataset (n), number of items of interest (ℓ), and the distribution and magnitude of the demands (Q). In our experiments, we evaluate the performance

of our 2-approximation and $(2+\epsilon)$-approximation algorithms, and compare them against two baselines: 1) the greedy approach and 2) RR-LP, the classical LP formulation for covering problems followed by a randomized rounding scheme. We also compare our main results against the exact DP algorithm in the technical report [51]. Across all settings, each experiment is repeated 30 times, and the reported results correspond to the average values.

5.2 Experiment Setting

5.2.1 Setup. The experiments were conducted on an Apple M2 Pro processor, 16 GB memory, running macOS. The algorithms were implemented in Python 3.12. We use Gurobi [26] LP solver in the implementation of our algorithms.[3]

5.2.2 Datasets. To evaluate our algorithms in practical settings, we employ the following four real-world datasets:

- Census [38]: This dataset contains a one percent sample of the *Public Use Microdata Samples* person records derived from the complete 1990 census dataset. It includes around 2.5 million records with 68 categorical attributes. We preprocess the dataset by transforming each attribute value into a distinct binary attribute. Additionally, we assign a randomly generated weight between 1 and 1000 to each tuple.
- Music [16]: This dataset is part of the Magellan data repository and comprises approximately 56,000 songs from the Amazon Music dataset. Each song is associated with a set-valued attribute called Genre. We used the Genre attribute to represent the sets in our problem. The duration of each song, measured in seconds, is used as the weight associated with the corresponding tuple. Notably, these weights are fairly similar across entries.
- Stack Overflow [56]: This dataset is derived from the 2024 Stack Overflow Annual Survey of Developers and contains rich categorical information on technologies, roles, education, and demographics. We selected 15 attributes from the dataset and transformed them into 114 binary attributes. Each tuple is additionally assigned a randomly generated weight between 1 and 1000.
- Yelp [29]: The Yelp business dataset contains approximately 150,000 business entries. From the Categories attribute, we extract items and generate distinct binary attributes corresponding to the unique category values. We then select the 90 most frequent attributes across all entries. Additionally, we use the Number of Reviews column as the weight attribute.

5.2.3 Parameter Defaults and Ranges. The default demand distribution is defined as a randomly generated integer uniformly sampled from the range 1 to 10. In the experiments evaluating the impact of demands, we generate each group's demands uniformly at random within a specified window, and we increase the window range in successive powers of two, ranging from 2^0 to 2^{12}. In cases where the generated demand cannot be satisfied by the data, we assign the item its maximum feasible demand value (the total number of sets that contain this item). The default setting assigns the number of items ℓ to 20 and the number of sets n to 50,000. The parameter ϵ in our $(2+\epsilon)$-approximation algorithm is fixed to 0.2 across all settings. In experiments assessing the impact of n, the number of sets is varied from 1,024 up to the full size of the dataset. In experiments evaluating the impact of ℓ, the number of items is varied from 3 up to the total number of items present in the dataset.

5.3 Implementation

5.3.1 Baselines. There is relatively little prior work directly addressing the special case of the weighted set multi-cover problem we studied in this paper. To the best of our knowledge, the greedy

[3]Code available at: https://github.com/neemashahbazi/WSMC-BU.

algorithm is the main practical baseline, as it is the most commonly used algorithm in this area. As another baseline, we also evaluate the classical randomized rounding algorithm used for the set cover problem and its variants. Further details for each of the two baselines are discussed below.

Greedy: The greedy baseline extends the well-known approximation algorithm for the unweighted Set Cover problem, which at each step selects the set covering the largest number of uncovered elements, achieving an $O(\log(\ell))$ approximation. Dobson [19] generalized this approach to the weighted Set Multi-Cover problem by introducing a cover-to-weight ratio, defined as the number of elements with positive residual demand in a set divided by its weight, and iteratively selecting the set with the highest ratio. A straightforward implementation requires $O(\ell n^2)$ time, as ratios must be recomputed for all remaining sets at each step. By using a priority queue and updating only the necessary values, the running time can be reduced to $O(\ell n \log(n))$, which becomes near-linear $O(n \log(n))$ when the number of items ℓ is constant, while preserving the same $O(\log(\ell)) = O(1)$ approximation guarantee. We note that, indeed, for the sake of a fair comparison, our greedy baseline is implemented using the faster approach.

RR-LP: The RR-LP baseline implements the classical LP formulation for the covering problems, followed by a randomized rounding scheme. In this approach, for each set $t \in \mathcal{D}$, a variable x_t is introduced, and the LP formulation is as follows:

$$\begin{aligned} \text{minimize:} \quad & \sum_{t \in \mathcal{D}} w_t \cdot x_t \\ \text{subject to:} \quad & \sum_{t \in \mathcal{D}, g \in t} x_t \geq Q_g, \quad \forall g \in \mathcal{G}, \\ & 0 \leq x_t \leq 1, \quad \forall t \in \mathcal{D}. \end{aligned} \tag{15}$$

After solving the above LP, each tuple $t \in \mathcal{D}$ is selected independently with probability $\min\{1, \log(\ell) \cdot x_t^*\}$, where x_t^* is the value of x_t in the optimal solution. Repeating the rounding a sufficient number of times and returning the best solution obtained yields an $O(\log \ell)$-approximation for WSMC-BU [59]. The runtime of this baseline is dominated by solving the LP and is therefore $\mathcal{L}(n)$.

5.3.2 Our algorithms. We implement the two algorithms described in Sections 4.1 and 4.2. Once the LP instance is solved, we round all values down and compute the residual demands that remain after rounding. The final solution is then obtained by adding extra tuples as specified in Algorithm 1. In the rare event that the total residual demand is large after the rounding, we replace this procedure with a greedy algorithm to fulfill the remaining item demands. The key distinction between the 2-approximation algorithm from Section 4.1 and the $(2+\epsilon)$-approximation algorithm from Section 4.2 lies in the significantly smaller LP instance generated and solved in the latter. To assess the impact of the optimizations introduced in the $(2+\epsilon)$-approximation algorithm on execution time, we evaluate both algorithms in the experiments. However, the $(2+\epsilon)$-approximation algorithm is considered our main result. In the technical report [51], we evaluate the exact DP algorithm. Since the DP algorithm is computationally expensive, the related experiments are evaluated over smaller data as described in [51], and the primary goal is to assess the quality of the solutions produced by our approaches against the best possible (optimal) solution.

5.4 Results

In this section, we present the results and key findings from our experiments on real datasets, compared to the two baselines.

5.4.1 Effectiveness. We begin by evaluating the effectiveness of the algorithms as the number of items increases. The results for the different datasets are presented in Figures 5, 6, 7, and 8.

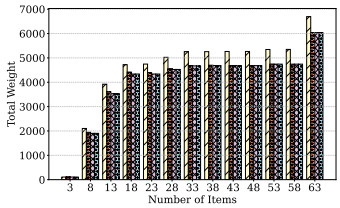


Fig. 5. Impact of varying number of items ℓ on the solution's total weight, Census

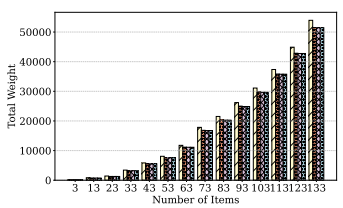


Fig. 6. Impact of varying number of items ℓ on the solution's total weight, Music

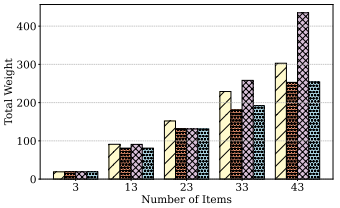


Fig. 7. Impact of varying number of items ℓ on the solution's total weight, Stack Overflow

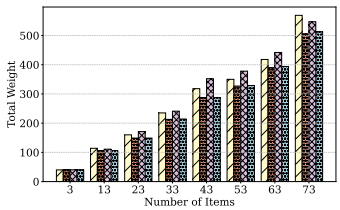


Fig. 8. Impact of varying number of items ℓ on the solution's total weight, Yelp

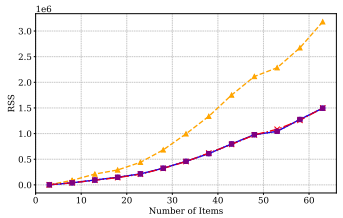


Fig. 9. Impact of varying number of items ℓ on deviation from demands, Census

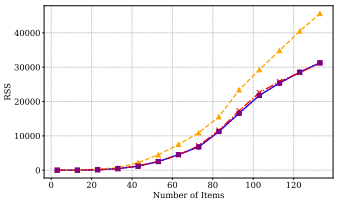


Fig. 10. Impact of varying number of items ℓ on deviation from demands, Music

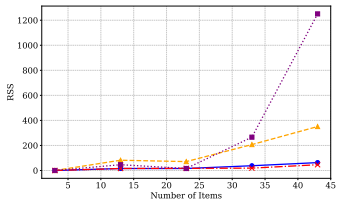


Fig. 11. Impact of varying number of items ℓ on deviation from demands, Stack Overflow

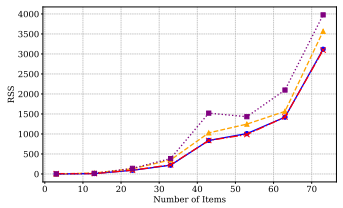


Fig. 12. Impact of varying number of items ℓ on deviation from demands, Yelp

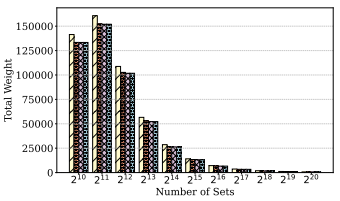


Fig. 13. Impact of varying number of sets n on the solution's total weight, Census

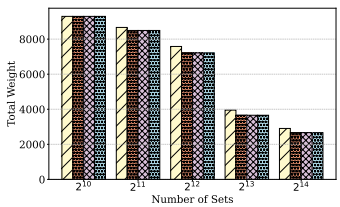


Fig. 14. Impact of varying number of sets n on the solution's total weight, Music

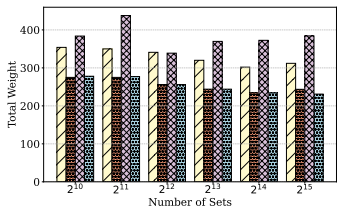


Fig. 15. Impact of varying number of sets n on the solution's total weight, Stack Overflow

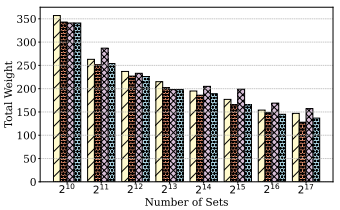


Fig. 16. Impact of varying number of sets n on the solution's total weight, Yelp

The key observation is that the solution weight increases as the number of items grows. This is intuitive, since a larger number of items requires selecting more sets to satisfy all demands, thereby increasing the total weight. Our algorithms consistently outperform or match the two baselines across all datasets while largely coinciding with each other. Notably, as shown in the technical report [51], our results closely align with those obtained by the exact DP algorithm.

Then, we evaluate how much each approach deviates from the requested demands as the number of items increases. This deviation is measured using the residual sum of squares (RSS), defined as

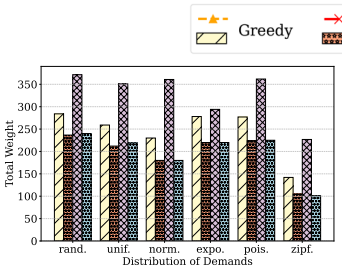


Fig. 17. Impact of demands distribution on the solution's total weight, Stack Overflow

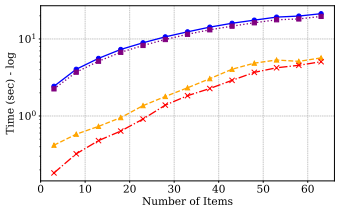


Fig. 18. Impact of varying number of items ℓ on the running time, Census

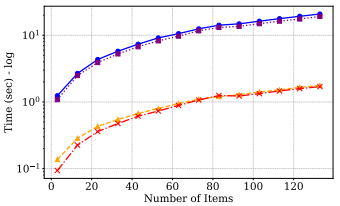


Fig. 19. Impact of varying number of items ℓ on the running time, Music

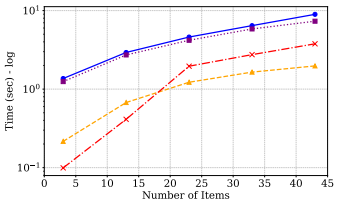


Fig. 20. Impact of varying number of items ℓ on the running time, Stack Overflow

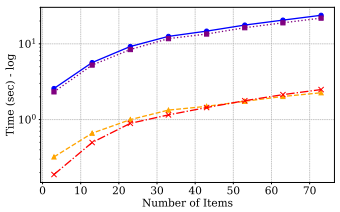


Fig. 21. Impact of varying number of items ℓ on the running time, Yelp

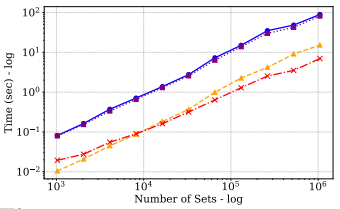


Fig. 22. Impact of varying number of sets n on the running time, Census

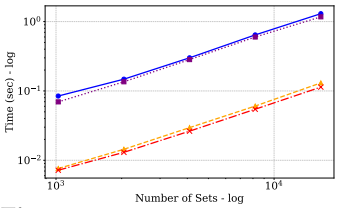


Fig. 23. Impact of varying number of sets n on the running time, Music

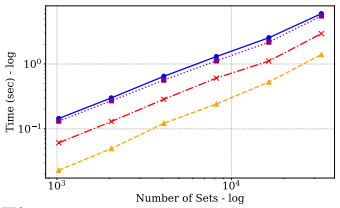


Fig. 24. Impact of varying number of sets n on the running time, Stack Overflow

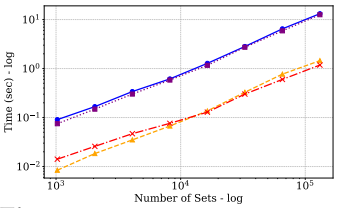


Fig. 25. Impact of varying number of sets n on the running time, Yelp

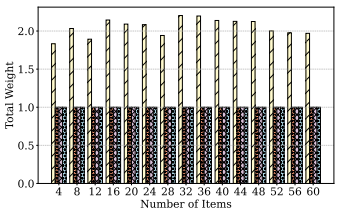


Fig. 26. Impact of varying number of items ℓ on the solution's total weight, Synthetic

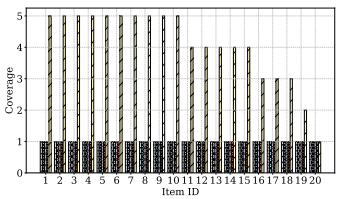


Fig. 27. Comparison of the actual demands vs. the item coverage, Synthetic, $\ell = 20^4$.

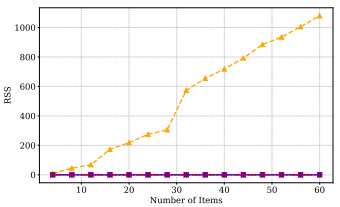


Fig. 28. Impact of varying number of items ℓ on deviation from demands, Synthetic

the sum of squared differences between the original demands and the covered demands across all items. An example demonstrating why this is an important measure for fairness applications of our work is given in the technical report [51]. The results are presented in Figures 9, 10, 11, and 12. The results indicate that the greedy algorithm consistently exhibits higher RSS, reflecting its tendency to select more sets than needed from each item compared to our algorithms. We also observe that RR-LP sometimes matches our algorithms but significantly underperforms in others.

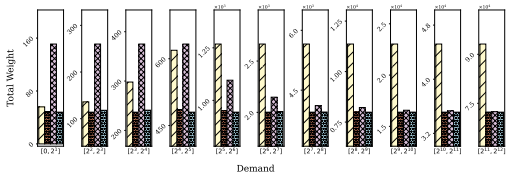


Fig. 29. Impact of varying demands on the solution's total weight, Stack Overflow

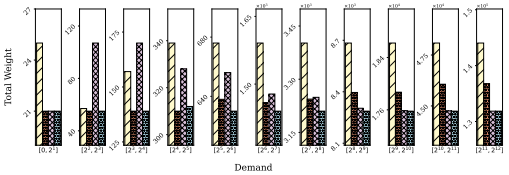


Fig. 30. Impact of varying demands on the solution's total weight, Yelp

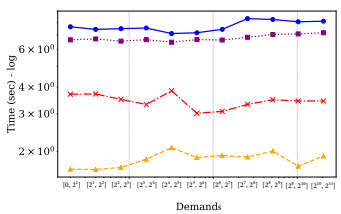


Fig. 31. Impact of varying demands on the running time, Stack Overflow

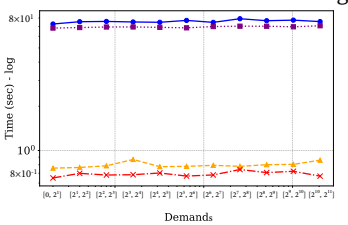


Fig. 32. Impact of varying demands on the running time, Yelp

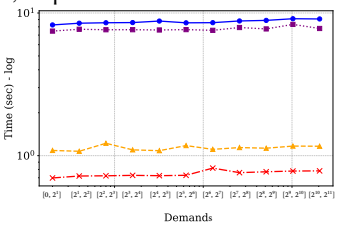


Fig. 33. Impact of varying demands on the running time, Census

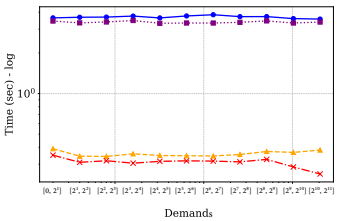


Fig. 34. Impact of varying demands on the running time, Music

In the next experiment, we examine the impact of varying the number of sets n on the total weight of the solution for each algorithm, noting that the objective is to select the solution with the lowest weight while satisfying the demands. The results for this experiment are illustrated in Figures 13, 14, 15 and 16. The first key observation is that the greedy baseline consistently underperforms compared to our two algorithms. In some cases, the Greedy baseline returns a solution with total weight 1.3× larger than the total weight returned by our algorithms. RR-LP, as in the previous experiment, can either match or fall short. Since our two algorithms are essentially the same—one being a faster variant of the other—they produce nearly identical results across most settings. In a few rare instances, the $(2+\epsilon)$ algorithm achieves slightly higher quality solutions, though the improvement is marginal and practically negligible; in other cases, it performs slightly worse, reflecting the influence of ϵ. The next observation is a consistent decreasing trend in the total weight as n increases across different datasets. This is primarily because additional sets provide more opportunities to satisfy the demands at lower cost, leading to lower-weight solutions. In the case of Census shown in Figure 13, the initial increasing trend arises because the demands cannot be satisfied if all the demands are generated at random, and hence some of them are manipulated as described in Section 5.2.3, to keep the problem feasible.

Next, we examine the impact of demand distributions on the effectiveness of our algorithms. Specifically, we generate demands according to six different distributions: Random, Uniform, Normal, Exponential, Poisson, and Zipfian. Owing to space constraints and the similarity of the results across datasets, we only report the results for the Stack Overflow dataset. The distribution parameters are defined as follows: Random: $Q \sim \text{Uniform}(1, 10)$, Uniform: $Q = 5$, Normal: $Q \sim \mathcal{N}(\mu = 5, \sigma^2 = 4)$,

Exponential: $Q \sim \text{Exp}(\lambda = \frac{1}{5})$, Poisson: $Q \sim \text{Poisson}(\lambda = 5)$, Zipfian: $Q \sim \text{Zipf}(s = 2)$. Our findings indicate that the solution weight is largely oblivious to the choice of demand distribution, as shown in Figure 17. Finally, we study the impact of demand magnitude on the effectiveness of our algorithms. To this end, we generate demand values uniformly at random within a specified range and vary this range from small to large values as previously explained. The results for two of the datasets are shown in Figures 29 and 30. Due to space constraints, the results for the other two datasets are shown in the technical report [51], and they follow a similar pattern. It can be observed that as demand values increase, the total weight of the solution increases, since the solution needs to choose more sets to satisfy the greater demands. Moreover, the results show that regardless of demand magnitude, our approaches consistently outperform the greedy baseline in terms of solution quality. The comparison between the greedy algorithm and our approaches seems to remain consistent regardless of demand size. In contrast, while the RR-LP baseline generally produces weaker solutions than our algorithms, its solution's quality improves as demands increase. For very large demand values, RR-LP approaches the quality of our methods, but it remains consistently worse than our 2-approximation algorithm. In some instances with large demands, RR-LP outperforms our $(2+\epsilon)$-approximation algorithm; however, RR-LP is always slower by multiple orders of magnitude, as shown in Figures 31, 32, 33, and 34.

Overall, in every experiment, both of our algorithms outperform the baselines in solution quality, with particularly consistent superiority over the widely used greedy approach. Although the differences may not always be large, it is important to note that the weights in these experiments are mainly random, which works against our approach. In some real-world scenarios, where weights are more likely to correlate meaningfully with the number of items a set contains, this performance gap could be significantly larger. An example of such cases is discussed in Section 5.5, showing the large gap between the quality of results returned by the greedy algorithm and our algorithms. As a rule of thumb, when solution quality is crucial, our approaches represent the only viable option.

5.4.2 Efficiency. We next turn our attention to the efficiency of our algorithms and compare them with the two baselines. We begin by examining the impact of increasing the number of items on running time. The results are shown in Figures 18, 19, 20, and 21. The first observation is that the 2-approximation algorithm and the RR-LP baseline are the slowest, with running times roughly an order of magnitude higher than the other two. The $(2+\epsilon)$-approximation algorithm again closely matches the efficiency of the greedy algorithm.

Next, we examine how increasing the number of sets affects the running time. The results are presented in Figures 22, 23, 24, and 25. As the number of sets increases, the running time of all algorithms grows. Similar to the previous experiment, our 2-approximation algorithm, followed by RR-LP, are the slowest among the four methods. The $(2+\epsilon)$-approximation algorithm closely matches the efficiency of the greedy; however, the greedy algorithm typically exhibits a steeper growth rate, leading to slightly lower running times for smaller values of n but higher running times as n increases.

This demonstrates the scalability of our algorithm. In particular, our optimization for the $(2+\epsilon)$-approximation proves highly effective, achieving a substantial reduction in running time—comparable to the greedy approach—while delivering superior solution quality.

Finally, we evaluate the impact of varying the magnitude of demands on the efficiency of the algorithms. The results, shown in Figures 31-34, confirm that the performance of all approaches is almost independent of the demand values over all the datasets.

Summary of results: Our main result—the $(2+\epsilon)$-approximation algorithm—consistently outperforms the baselines across all settings, in terms of effectiveness, efficiency, or both. When

effectiveness is comparable to that of the baselines, our approach holds a clear advantage in efficiency. Conversely, when efficiency is similar, our method surpasses the baselines in effectiveness. While the running time of our algorithm is comparable to that of the greedy baseline, it consistently delivers higher solution quality. Conversely, although its solution quality is similar to the RR-LP baseline and the 2-approximation algorithm, our method achieves substantially better running time. In summary, our algorithm offers the best of both worlds, combining high solution quality with efficiency.

5.5 Case Study: An Adversarial Instance

Finally, we highlight a scenario where the weight of the sets is proportional to the number of items they include and demonstrate that our algorithms substantially outperform the greedy baseline in terms of the solution quality. For example, the expected salary of software engineers can be correlated with the number of programming languages they know. The primary goal of these experiments is to highlight the fact that in some scenarios, the quality of the solution produced by the greedy baseline can be significantly worse than what our algorithms achieve. To this end, we construct a synthetic dataset with ℓ items and $n = \ell$ sets. The sets are constructed such that the first set contains only the first item, the second set contains the first and second items, and so forth, until the ℓ-th set, which includes all items. Additionally, we assign weights to the sets as follows: the first set is assigned a weight of $\frac{1}{\ell}$, the second set a weight of $\frac{1}{\ell-1}$, the third set a weight of $\frac{1}{\ell-2}$, continuing in this manner up to the $\ell - 1$-th set. We set the weight of the ℓ-th set to 1.01. We further set the demand of each item to 1. We begin by examining the effect of increasing the number of items (both ℓ and n in this case) on the total solution weight. As illustrated in Figure 26, the total weight of the sets selected by the greedy baseline is up to twice that of the solution produced by our algorithms. This is because the greedy algorithm selects sets based on the best cover-to-weight ratio, which implies that the ℓ-th set is not chosen in the early stages. This not only leads to a suboptimal solution in terms of total weight but also causes the greedy solution to deviate unnecessarily from the required demands by up to a factor of five, as illustrated in Figure 27. In contrast, both of our algorithms and the RR-LP baseline select only the ℓ-th set as the solution, which in this case is optimal and satisfies all demands without deviation. This further becomes clear by the measured RSS between the required and covered demands, as illustrated in Figure 28.

6 Related Work

The *set cover* problem is one of the most extensively studied problems in computer science. The classical greedy algorithm achieves an $O(\log \ell)$-approximation [15], which is asymptotically optimal [18]. Numerous variations of set cover have also been explored, including partial set cover [46], set multi-cover problem [19], and multi-set multi-cover problem [28]. Generally, $O(\log \ell)$-approximation algorithms are known for broad classes of covering and packing problems via integer programming methods [11, 32, 55].

For the *set multi-cover problem* (unbounded universe), which generalizes the WSMC-BU problem, the greedy algorithm of Dobson [19] achieves an $O(\log \ell)$-approximation, with the same guarantee holding in the weighted case. In geometric settings, where the universe $\mathcal{G}$ consists of points in $\mathbb{R}^d$ (for constant d) and sets correspond to geometric objects such as balls or halfspaces, $O(\log \mathrm{OPT})$- or even constant-factor approximations are known for both the unweighted [8] and weighted cases [9, 45].

Specifically for the WSMC-BU problem, Bredereck et al. [4] provide an exact algorithm running in $O(h(\ell) \cdot \mathrm{poly}(n)) = O(\mathrm{poly}(n))$ time, where $h(\ell)$ is a function exponential on ℓ, and $\mathrm{poly}(n)$ is a

[4]Smaller ℓ selected for clarity of visualization.

polynomial function with respect to n. However, this approach is of limited practical use, as the polynomial factors are not explicitly analyzed and the constants involved appear prohibitively large. On the other hand, the greedy algorithm for weighted set multi-cover (unbounded universe) [19] directly yields an $O(\log \ell) = O(1)$-approximation for the WSMC-BU problem in $O(n \log n)$ time.

A closely related problem is *maximum coverage*, where the goal is to select k sets from $\mathcal{D}$ that maximize the number of covered elements. The greedy algorithm guarantees a $(1 - \frac{1}{e})$-approximation [43]. Improved algorithms exist for set systems of bounded VC-dimension [2]. Several extensions have been investigated, including maximum coverage with constraints [10], the maximum multi-coverage problem [3], and the minimum p-union problem [47].

Several problems studied in the database community can be naturally modeled as instances of WSMC-BU and can therefore benefit from our algorithms. Below, we briefly discuss two representative application domains and focus on how prior work differs from our setting.

Crowd-sourcing and task assignment. A large body of work in databases formulates crowd-sourcing and task-assignment problems as selecting a set of workers, each associated with a cost and a set of skills, to satisfy coverage requirements over tasks or skill types [35, 58, 62, 64–66]. These formulations correspond to weighted set multi-cover instances, where workers define sets, skills, or task types define universe elements, and demands capture the required number of assignments. In many practical settings, however, the number of skills or task types is small (e.g., domains in Example 2 or programming languages in Example 4), making these problems well-suited for our WSMC-BU framework.

Fairness-aware data selection. Fairness-aware data management and ML pipelines often require selecting a subset of data that satisfies representation constraints over protected attributes [1, 12, 13, 22, 33, 36, 39, 42, 53, 57, 63]. Such problems can be modeled as weighted set multi-cover instances and variations of it. The number of protected attributes is usually small, which naturally aligns these problems with the bounded-universe setting of WSMC-BU. For example, there are 5 standard categories for gender and 7 standard categories for ethnicity in Example 1.

We emphasize that these are representative examples. Due to the generality of the set cover framework, WSMC-BU also applies more broadly to other data-driven decision-making tasks such as constrained recommendation problems [44, 50].

Most existing approaches in crowd-sourcing, fairness-aware data selection, and package recommendation employ greedy heuristics or LP-based techniques for covering or packing problems [22, 50, 52]. Some of these approaches [22, 52] can be directly formulated as instances of WSMC-BU, while others consider closely related variants with different objectives or constraints [50, 62]. Independent of these modeling choices, none of the existing methods exploits a bounded universe size. To the best of our knowledge, we are the first to study the *weighted set multi-cover* problem on a bounded universe within the database community. While this problem has been studied in the theory community [4], no efficient 2-approximation (or better) algorithm was previously known. Our approximation algorithms crucially leverage the bounded universe size to achieve strictly better approximation guarantees for WSMC-BUwhile remaining efficient. As a result, our approach is fundamentally different from both the classical greedy algorithm and standard LP-based solutions. For instance, although the LP formulation in LP 10 resembles standard formulations such as [19], we introduce a novel rounding technique that is efficient only under the bounded-universe assumption. Furthermore, we establish a non-trivial connection to Problem 9, which enables us to accelerate the algorithm in Section 4.2 by approximating its objective function with a sum of piecewise-linear functions.

7 Conclusion

In this paper, we study the WSMC-BU problem and propose three algorithms: an exact dynamic programming algorithm, a 2-approximation algorithm, and a more efficient $(2+\varepsilon)$-approximation algorithm. Our methods outperform the Greedy and the standard LP-rounding baselines both theoretically (achieving better approximation guarantees with faster running times) and empirically, as demonstrated through experiments on real and synthetic datasets. Several interesting directions remain open. A key question is whether a polynomial-time $(1+\varepsilon)$-approximation algorithm exists for the WSMC-BU problem, for any $\varepsilon > 0$. Another interesting problem is to study the WSMC-BU problem in streaming or dynamic settings. We believe our results provide a solid foundation for exploring these extensions in the future.

References

[1] Abolfazl Asudeh, Tanya Berger-Wolf, Bhaskar DasGupta, and Anastasios Sidiropoulos. 2023. Maximizing coverage while ensuring fairness: A tale of conflicting objectives. *Algorithmica* 85, 5 (2023), 1287–1331.

[2] Ashwinkumar Badanidiyuru, Robert Kleinberg, and Hooyeon Lee. 2012. Approximating low-dimensional coverage problems. In *Proceedings of the twenty-eighth annual symposium on Computational geometry*. 161–170.

[3] Siddharth Barman, Omar Fawzi, Suprovat Ghoshal, and Emirhan Gürpınar. 2022. Tight approximation bounds for maximum multi-coverage. *Mathematical Programming* 192, 1 (2022), 443–476.

[4] Robert Bredereck, Piotr Faliszewski, Rolf Niedermeier, Piotr Skowron, and Nimrod Talmon. 2020. Mixed integer programming with convex/concave constraints: Fixed-parameter tractability and applications to multicovering and voting. *Theoretical Computer Science* 814 (2020), 86–105.

[5] Matteo Brucato, Azza Abouzied, and Alexandra Meliou. 2017. A scalable execution engine for package queries. *ACM SIGMOD Record* 46, 1 (2017), 24–31.

[6] Matteo Brucato, Juan Felipe Beltran, Azza Abouzied, and Alexandra Meliou. 2016. Scalable Package Queries in Relational Database Systems. *Proceedings of the VLDB Endowment* 9, 7 (2016).

[7] Chengliang Chai, Guoliang Li, Jian Li, Dong Deng, and Jianhua Feng. 2016. Cost-Effective Crowdsourced Entity Resolution: A Partial-Order Approach. In *Proceedings of the 2016 International Conference on Management of Data* (San Francisco, California, USA) *(SIGMOD '16)*. Association for Computing Machinery, New York, NY, USA, 969–984. doi:10.1145/2882903.2915252

[8] Chandra Chekuri, Kenneth L Clarkson, and Sariel Har-Peled. 2012. On the set multicover problem in geometric settings. *ACM Transactions on Algorithms (TALG)* 9, 1 (2012), 1–17.

[9] Chandra Chekuri, Sariel Har-Peled, and Kent Quanrud. 2020. Fast LP-based approximations for geometric packing and covering problems. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*. SIAM, 1019–1038.

[10] Chandra Chekuri and Amit Kumar. 2004. Maximum coverage problem with group budget constraints and applications. In *International Workshop on Randomization and Approximation Techniques in Computer Science*. Springer, 72–83.

[11] Chandra Chekuri and Kent Quanrud. 2019. On approximating (sparse) covering integer programs. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*. SIAM, 1596–1615.

[12] Zhao Chen, Peng Cheng, Lei Chen, Xuemin Lin, and Cyrus Shahabi. 2020. Fair task assignment in spatial crowdsourcing. *Proceedings of the VLDB Endowment* 13, 12 (2020).

[13] Peng Cheng, Xun Jian, and Lei Chen. 2018. An experimental evaluation of task assignment in spatial crowdsourcing. *Proceedings of the VLDB Endowment* 11, 11 (2018), 1428–1440.

[14] Dong-Yeon Cho, Yoo-Ah Kim, and Teresa M Przytycka. 2012. Chapter 5: Network biology approach to complex diseases. *PLoS computational biology* 8, 12 (2012), e1002820.

[15] Vasek Chvatal. 1979. A greedy heuristic for the set-covering problem. *Mathematics of operations research* 4, 3 (1979), 233–235.

[16] Sanjib Das, AnHai Doan, Paul Suganthan G. C., Chaitanya Gokhale, Pradap Konda, Yash Govind, and Derek Paulsen. [n. d.]. The Magellan Data Repository. https://sites.google.com/site/anhaidgroup/useful-stuff/the-magellan-data-repository.

[17] Mohsen Dehghankar, Rahul Raychaudhury, Stavros Sintos, and Abolfazl Asudeh. 2025. Fair set cover. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V. 1*. 189–200.

[18] Irit Dinur and David Steurer. 2014. Analytical approach to parallel repetition. In *Proceedings of the forty-sixth annual ACM symposium on Theory of computing*. 624–633.

[19] Gregory Dobson. 1982. Worst-case analysis of greedy heuristics for integer programming with nonnegative data. *Mathematics of Operations Research* 7, 4 (1982), 515–531.

[20] Xin Dong, Evgeniy Gabrilovich, Geremy Heitz, Wilko Horn, Ni Lao, Kevin Murphy, Thomas Strohmann, Shaohua Sun, and Wei Zhang. 2014. Knowledge vault: a web-scale approach to probabilistic knowledge fusion. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (New York, New York, USA) *(KDD '14)*. Association for Computing Machinery, New York, NY, USA, 601–610. doi:10.1145/2623330.2623623

[21] Marina Drosou and Evaggelia Pitoura. 2012. DisC diversity: result diversification based on dissimilarity and coverage. *Proceedings of the VLDB Endowment* 6, 1 (2012), 13–24.

[22] M. Erfanian et al. 2024. Chameleon: Foundation Models for Fairness-Aware Multi-Modal Data Augmentation to Enhance Coverage of Minorities. *Proc. VLDB Endow.* 17, 11 (July 2024), 3470–3483. doi:10.14778/3681954.3682014

[23] Michael J. Franklin, Donald Kossmann, Tim Kraska, Sukriti Ramesh, and Reynold Xin. 2011. CrowdDB: answering queries with crowdsourcing. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data* (Athens, Greece) *(SIGMOD '11)*. Association for Computing Machinery, New York, NY, USA, 61–72. doi:10.1145/1989323.1989331

[24] Michael J. Franklin, Beth Trushkowsky, Purnamrita Sarkar, and Tim Kraska. 2013. Crowdsourced enumeration queries. In *Proceedings of the 2013 IEEE International Conference on Data Engineering (ICDE 2013) (ICDE '13)*. IEEE Computer Society, USA, 673–684. doi:10.1109/ICDE.2013.6544865

[25] Xun Ge. 2010. An application of covering approximation spaces on network security. *Computers & Mathematics with Applications* 60, 5 (2010), 1191–1199.

[26] Gurobi Optimization, LLC. 2025. Gurobi Optimizer Reference Manual. https://www.gurobi.com

[27] Karla L Hoffman and Manfred Padberg. 1993. Solving airline crew scheduling problems by branch-and-cut. *Management science* 39, 6 (1993), 657–682.

[28] Qiang-Sheng Hua, Dongxiao Yu, Francis CM Lau, and Yuexuan Wang. 2009. Exact algorithms for set multicover and multiset multicover problems. In *International Symposium on Algorithms and Computation*. Springer, 34–44.

[29] Yelp Inc. 2025. Yelp Open Dataset. https://business.yelp.com/data/resources/open-dataset/. Accessed: 2025-08-20.

[30] Shunhua Jiang, Zhao Song, Omri Weinstein, and Hengjie Zhang. 2021. A faster algorithm for solving general LPs. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*. 823–832.

[31] Aniket Kittur, Ed H. Chi, and Bongwon Suh. 2008. Crowdsourcing user studies with Mechanical Turk. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Florence, Italy) *(CHI '08)*. Association for Computing Machinery, New York, NY, USA, 453–456. doi:10.1145/1357054.1357127

[32] Stavros G Kolliopoulos and Neal E Young. 2005. Approximation algorithms for covering/packing integer programs. *J. Comput. System Sci.* 71, 4 (2005), 495–505.

[33] Yash Kurkure, Miles Shamo, Joseph Wiseman, Sainyam Galhotra, and Stavros Sintos. 2024. Faster algorithms for fair max-min diversification in rd. *Proceedings of the ACM on Management of Data* 2, 3 (2024), 1–26.

[34] Jon Lee and Dan Wilson. 2001. Polyhedral methods for piecewise-linear functions I: the lambda method. *Discrete applied mathematics* 108, 3 (2001), 269–285.

[35] Guoliang Li, Chengliang Chai, Ju Fan, Xueping Weng, Jian Li, Yudian Zheng, Yuanbing Li, Xiang Yu, Xiaohang Zhang, and Haitao Yuan. 2017. CDB: Optimizing Queries with Crowd-Based Selections and Joins. In *Proceedings of the 2017 ACM International Conference on Management of Data (SIGMOD)*. 1463–1478.

[36] Yifan Li, Xiaohui Yu, and Nick Koudas. 2021. Data acquisition for improving machine learning models. *Proceedings of the VLDB Endowment* 14, 10 (2021), 1832–1844.

[37] Helena R Lourenço, José P Paixão, and Rita Portugal. 2001. Multiobjective metaheuristics for the bus driver scheduling problem. *Transportation science* 35, 3 (2001), 331–343.

[38] Chris Meek, Bo Thiesson, and David Heckerman. 2001. US Census Data (1990). https://archive.ics.uci.edu/dataset/116/us+census+data+1990. UCI Machine Learning Repository. DOI: https://doi.org/10.24432/C5VP42.

[39] Melika Mousavi, Nima Shahbazi, and Abolfazl Asudeh. 2024. Data Coverage for Detecting Representation Bias in Image Datasets: A Crowdsourcing Approach. In *International Conference on Extending Database Technology (EDBT)*.

[40] Kamesh Munagala, Shivnath Babu, Rajeev Motwani, and Jennifer Widom. 2005. The pipelined set cover problem. In *International Conference on Database Theory*. Springer, 83–98.

[41] Ashwin Narayan, Vuk Marković, Natalia Postawa, Anna King, Alejandro Morales, K Ashwin Kumar, and Petros Efstathopoulos. 2016. Efficient routing for cost effective scale-out data architectures. In *2016 IEEE 24th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS)*. IEEE, 424–429.

[42] Fatemeh Nargesian, Abolfazl Asudeh, and H. V. Jagadish. 2021. Tailoring data source distributions for fairness-aware data integration. 14, 11 (July 2021), 2519–2532. doi:10.14778/3476249.3476299

[43] George L Nemhauser, Laurence A Wolsey, and Marshall L Fisher. 1978. An analysis of approximations for maximizing submodular set functions—I. *Mathematical programming* 14, 1 (1978), 265–294.

[44] Shuyao Qi, Nikos Mamoulis, Evaggelia Pitoura, and Panayiotis Tsaparas. 2016. Recommending packages to groups. In *2016 IEEE 16th International Conference on Data Mining (ICDM)*. IEEE, 449–458.
[45] Rajiv Raman and Saurabh Ray. 2020. Improved approximation algorithm for set multicover with non-piercing regions. In *28th Annual European Symposium on Algorithms (ESA 2020)*. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 78–1.
[46] Yingli Ran, Xiaohui Huang, Zhao Zhang, and Ding-Zhu Du. 2022. Approximation algorithm for minimum partial multi-cover under a geometric setting. *Optimization Letters* 16, 2 (2022), 667–680.
[47] Yingli Ran and Zhao Zhang. 2022. Approximation Algorithm for Minimum p Union Under a Geometric Setting. *arXiv preprint arXiv:2208.14264* (2022).
[48] Charles ReVelle, Constantine Toregas, and Louis Falkson. 1976. Applications of the location set-covering problem. *Geographical analysis* 8, 1 (1976), 65–76.
[49] Jerrold Rubin. 1973. A technique for the solution of massive set covering problems, with application to airline crew scheduling. *Transportation Science* 7, 1 (1973), 34–48.
[50] Dimitris Serbos, Shuyao Qi, Nikos Mamoulis, Evaggelia Pitoura, and Panayiotis Tsaparas. 2017. Fairness in package-to-group recommendations. In *Proceedings of the 26th international conference on world wide web*. 371–379.
[51] Nima Shahbazi, Aryan Esmailpour, and Stavros Sintos. 2026. Weighted Set Multi-Cover on Bounded Universe and Applications in Package Recommendation. https://arxiv.org/abs/2603.12528. arXiv:2603.12528 [cs.DS]
[52] Nima Shahbazi, Yin Lin, Abolfazl Asudeh, and HV Jagadish. 2023. Representation bias in data: a survey on identification and resolution techniques. *CSUR* (2023).
[53] Nima Shahbazi, Jin Wang, Zhengjie Miao, and Nikita Bhutani. 2024. Fairness-aware Data Preparation for Entity Matching. *ICDE* (2024).
[54] Victor S. Sheng, Foster Provost, and Panagiotis G. Ipeirotis. 2008. Get another label? improving data quality and data mining using multiple, noisy labelers. In *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (Las Vegas, Nevada, USA) *(KDD '08)*. Association for Computing Machinery, New York, NY, USA, 614–622. doi:10.1145/1401890.1401965
[55] Aravind Srinivasan. 1999. Improved approximation guarantees for packing and covering integer programs. *SIAM J. Comput.* 29, 2 (1999), 648–670.
[56] Stack Overflow. 2024. Stack Overflow Developer Survey 2024. https://survey.stackoverflow.co/2024/ Accessed: 2025-08-17.
[57] Julia Stoyanovich, Ke Yang, and HV Jagadish. 2018. Online set selection with fairness and diversity constraints. In *Proceedings of the EDBT Conference*.
[58] Yongxin Tong and Zimu Zhou. 2018. Dynamic task assignment in spatial crowdsourcing. *Sigspatial Special* 10, 2 (2018), 18–25.
[59] Vijay V Vazirani. 2010. *Approximation Algorithms*. Springer, Berlin, Germany.
[60] Rao R Vemuganti. 1998. Applications of set covering, set packing and set partitioning models: A survey. In *Handbook of Combinatorial Optimization: Volume1–3*. Springer, 573–746.
[61] Laurence A Wolsey and George L Nemhauser. 1999. *Integer and combinatorial optimization*. John Wiley & Sons.
[62] Xiang Yu, Guoliang Li, Yudian Zheng, Yan Huang, Songfan Zhang, and Fei Chen. 2018. Crowdota: An online task assignment system in crowdsourcing. In *2018 IEEE 34th International Conference on Data Engineering (ICDE)*. IEEE, 1629–1632.
[63] Luca Zecchini, Vasilis Efthymiou, Felix Naumann, Giovanni Simonini, et al. 2025. Deduplicated Sampling On-Demand. *Proceedings of the VLDB Endowment* 18, 8 (2025), 2482–2495.
[64] Yudian Zheng, Guoliang Li, and Reynold Cheng. 2016. Docs: a domain-aware crowdsourcing system using knowledge bases. *Proceedings of the VLDB Endowment* 10, 4 (2016), 361–372.
[65] Yudian Zheng, Guoliang Li, Yuanbing Li, Caihua Shan, and Reynold Cheng. 2017. Truth inference in crowdsourcing: Is the problem solved? *Proceedings of the VLDB Endowment* 10, 5 (2017), 541–552.
[66] Yudian Zheng, Jiannan Wang, Guoliang Li, Reynold Cheng, and Jianhua Feng. 2015. QASCA: A Quality-Aware Task Assignment System for Crowdsourcing Applications. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data (SIGMOD)*. 1031–1046.

Received October 2025; revised January 2026; accepted February 2026