Estimating Entity Importance via Counting Set Covers

Aristides Gionis
Yahoo! Research
gionis@yahoo-inc.com

Theodoros Lappas
Boston University
tlappas@cs.bu.edu

Evimaria Terzi
Boston University
evimaria@cs.bu.edu

ABSTRACT

The data-mining literature is rich in problems asking to assess the importance of entities in a given dataset. At a high level, existing work identifies important entities either by *ranking* or by *selection*. Ranking methods assign a score to every entity in the population, and then use the assigned scores to create a ranked list. The major shortcoming of such approaches is that they ignore the redundancy between high-ranked entities, which may in fact be very similar or even identical. Therefore, in scenarios where diversity is desirable, such methods perform poorly. Selection methods overcome this drawback by evaluating the importance of a group of entities *collectively*. To achieve this, they typically adopt a *set-cover* formulation, which identifies the entities in the minimum set cover as the important ones. However, this dichotomy of entities conceals the fact that, even though an entity may not be in the reported cover, it may still participate in many other optimal or near-optimal solutions. In this paper, we propose a framework that overcomes the above drawbacks by integrating the ranking and selection paradigms. Our approach assigns importance scores to entities based on both the *number* and the *quality* of set-cover solutions that they participate in. Our methodology applies to a wide range of applications. In a user study and an experimental evaluation on real data, we demonstrate that our framework is efficient and provides useful and intuitive results.

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Miscellaneous; G.2.1 [**Discrete Mathematics**]: Combinatorics—*Combinatorial algorithms, Counting problems*

Keywords

counting, importance sampling, set cover

1. INTRODUCTION

How can we identify a subset of important entities from a given population? This question arises in numerous application domains. For example, how can we select a small subset of reviews to show to a user from over 35,000 reviews hosted on `Amazon.com` for the Kindle Keyboard 3G? Or, as another example, how can we select a set of experts from a large population of skilled individuals registered on sites like `linkedin.com`, `odesk.com` and `guru.com`?

Existing data-mining methods address this type of questions via *entity-ranking* or *entity-selection* methods.

Entity-ranking methods assign a score to each entity and then report to the user the top-k entities with the highest scores. For example, review portals like `Yelp` or `Amazon` rank reviews based on user-submitted helpfulness votes. Such score-assigning schemes ignore the redundancy between the highly-scored entities. For example, the top-k reviews about a laptop may all comment on the battery life and the weight of the laptop, but may provide no information about the quality of its screen or other important features.

Entity-selection methods overcome this drawback by *collectively* evaluating *sets* of entities, and reporting the members of the highest-scoring sets as important. For example, review-selection methods select a small subset of reviews that collectively comment upon all the attributes of a particular product [16, 18, 23]. Similarly, expert-selection methods identify a set of skilled experts that can collectively perform a given task [2, 17]. That is, entity-selection methods identify the entities in the selected subset as important, and naïvely dismiss the rest as unimportant. Such a dichotomy conceals the fact that there may be entities not selected in the discovered solution, but which may participate in equally-good (or almost as good) solutions.

In this paper, we propose a framework that combines the entity-ranking and entity-selection paradigms and overcomes their respective drawbacks. Given a set of entities $\mathcal{C}$, our methods assign an importance score to entities based on the number of *high-quality* solutions that they participate. In particular, we focus on entity-selection problems, which are formalized as *minimum set-cover* problems [1, 14, 16, 17, 21, 23]. The common characteristic of all these formulations is that the input consists of two parts: (i) a universe $U = \{u_1, \ldots, u_n\}$ of items that need to be covered, and (ii) a collection of entities, where each entity is a subset of this universe, i.e., $\mathcal{C} = \{C_1, \ldots, C_m\}$ with $C_i \subseteq U$. Any subset $\mathcal{S}$ of entities whose union contains the entire universe U, i.e., $\cup_{C \in \mathcal{C}'} C = U$, is called a *set cover* or simply a *cover*. The data-mining task is typically mapped to the problem of finding the *minimum set cover*. The assumption in such formulations is that the entities that participate in the minimum cover compose an important subset. Some of the applications this formulation are the following.

Review selection: What are the best reviews to display for a given product? In this application, the universe U consists of all the attributes of the reviewed product. Each review $C \in \mathcal{C}$ comments only on a subset $C \subseteq U$ of the attributes. Given the limited attention-

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
KDD'12, August 12–16, 2012, Beijing, China.
Copyright 2012 ACM 978-1-4503-1462-6 /12/08 ...$15.00.

span of the users, the review-selection problem asks for a small set of reviews that comment on all the attributes of a product [16, 23].

Expert selection: Who are the best experts to select for executing a given task? In this application, the universe U consists of the skills required for a particular task. Each expert $C \in \mathcal{C}$ is represented by the set of his skills $C \subseteq U$. The team-formation problem asks for a small-cardinality team whose members have the required set of skills [2, 17].

The underlying thesis in this paper is that by simply selecting the entities in the minimum set cover and dismissing other set-cover solutions one ignores useful information about the landscape of the solution space. For example, our analysis of the `Guru` dataset[1] revealed that although there are more than 340 000 registered experts, there are only 200 unique combinations of skills. That is, every expert is (on average) identical with approximately 1700 other experts. This means that every set $\mathcal{S}$ of cardinality $|\mathcal{S}|$ is as good as $1700^{|\mathcal{S}|}$ other teams (on average). However, the set-cover formulation naïvely selects to report as important the members of only one of these subsets

In our work, we take into consideration the landscape of solutions by adopting a *counting framework*; instead of focusing on a single optimal solution, we count the number of high-quality set covers that each entity participates. We use the cardinality of a set cover as a measure of its quality; compact covers are preferable, since they lead to lower-cost solutions.

On a high level, we address the following problem.

PROBLEM 1. *We are given a universe of elements U and a set of entities $\mathcal{C}$ where each $C \in \mathcal{C}$ is a subset of U. The goal is to evaluate the importance of each entity by counting the number of compact set covers that it participates.*

In the context of reviews, solving Problem 1 will allow us to to evaluate the utility of each review in the covering of all the attributes of a given product. Intuitively, our methodology will assign high importance to reviews that cover a large number of a product's attributes, or comment on a small number of attributes that are rarely commented by other reviewers. In *expert-management* applications, the scores assigned by our approach will be driven by the number of skills that each individual can contribute, as well as the rarity of these skills. Intuitively, these are the individuals who play a key role in forming effective teams. While we use the domains of reviews and experts to motivate our work, our methodology is applicable to any setting that can be expressed in the terms of the set-covering formulation.

Contribution: Given a set system $X = \langle U, \mathcal{C} \rangle$, where U is a universe of elements and $\mathcal{C}$ is a set of entities defined as subsets of U, our paper proposes a general methodology that assigns importance scores to entities in $\mathcal{C}$. The score of each entity depends both on the number and the quality of the set covers that the entity participates. Enumerating all the set covers that an entity participates is a computationally intractable task. Here, we take advantage of the fact that we are only interested in the *number* of set covers an entity participates. Therefore, we develop practical counting schemes that allow us to accurately estimate this number in polynomial time. More specifically, we propose a counting algorithm that is based on Monte Carlo (MC) sampling. The algorithm builds on existing techniques for counting the number of satisfying assignments of DNF formulas [22]. We significantly extend this technique in the following ways: (i) we modify it so that it can efficiently estimate the number of set covers of *all* entities simultaneously; (ii) we show how we can incorporate weights that capture the size of set cover solutions; (iii) we show how to handle efficiently entities with multiple copies. Finally, our experimental evaluation shows the scalability of our algorithm, and the utility of our methodology in different application domains. Also, our user study demonstrates that our concept of importance is in agreement with human perception regarding important entities.

Roadmap: The rest of the paper is organized as follows. In Section 2 we present an overview of our framework, and in Section 3 we present our algorithms. We present our experimental results in Section 4 and we discuss the related work in Section 5. We conclude the paper with discussion and conclusions in Section 6.

2. THE COUNTING FRAMEWORK

In this paper, we consider problems for which the input consists of two parts: (i) a universe set $U = \{u_1, \ldots, u_n\}$ of items that need to be covered, and (ii) a collection of entities, where each entity is a subset of this universe, i.e., $\mathcal{C} = \{C_1, \ldots, C_m\}$ with $C_i \subseteq U$. We use the notation $X = \langle U, \mathcal{C} \rangle$ to represent such an input. We visualize this type of input using a bipartite graph; the nodes on the one side of the graph correspond to the elements of U, and the nodes of the other side correspond to the entities in $\mathcal{C}$. An edge connects a node $u \in U$ to node $C \in \mathcal{C}$ if $u \in C$. A bipartite-graph representation of input $X = \langle U, \mathcal{C} \rangle$ with $U = \{u_1, u_2, u_3, u_4\}$ and $\mathcal{C} = \{C_1, \ldots, C_5\}$ is shown in Figure 1(a).

DEFINITION 1. *For input $X = \langle U, \mathcal{C} \rangle$ we say that a subset $\mathcal{S}$ of the collection $\mathcal{C}$ ($\mathcal{S} \subseteq \mathcal{C}$) is a* set cover *or simply a* cover *of U if $U \subseteq \bigcup_{C \in \mathcal{S}} C$.*

For the rest of the paper, we will use $\mathbf{L}$ to represent all the subsets of $\mathcal{C}$ and $\mathbf{L}_{\text{sc}}$ to represent all possible set covers.

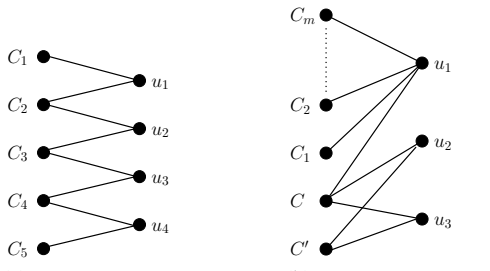
The minimum set cover. A set cover $\mathcal{S}^*$ that has minimum cardinality is called *a minimum set cover*. The problem of finding the minimum set cover, henceforth called SET-COVER, is a well-studied NP-hard problem, which can be approximated within a factor of $O(\log n)$ using a simple greedy algorithm [24].

In many application domains, a solution to SET-COVER is a natural way to select a small subset of $\mathcal{C}$: an entity C of the collection $\mathcal{C}$ is "selected" if and only if C is part of the solution $\mathcal{S}^*$. Then, the entities in $\mathcal{S}^*$ are rendered more "important" than the entities not in the solution. However, such a strict dichotomy can be unfair. For example, an entity C may be part of another minimum or near-minimum set cover. Such a case is shown in the following example.

EXAMPLE 1. *Consider the instance of the* SET-COVER *problem represented by the bipartite graph shown in Figure 1(a) with 4 items in U, 5 entities in $\mathcal{C}$, and $C_1 = \{u_1\}$, $C_2 = \{u_1, u_2\}$, $C_3 = \{u_2, u_3\}$, $C_4 = \{u_3, u_4\}$, and $C_5 = \{u_5\}$. In this case, the minimum set-cover solution is $\mathcal{C} = \{C_2, C_4\}$. The minimum-cardinality solution, in which set C_3 participates, has cardinality 3. Nevertheless, there is no reason to believe that set C_3 is so much less important than C_2 or C_4. After all, C_3 has as many elements as C_2 and C_4.*

Minimal set covers. Instead of focusing on a single minimum-cardinality solution, we could *count the number of solutions* that each entity $C \in \mathcal{C}$ participates. One way of applying this idea is to consider the concept of *minimal set covers*. A set cover $\mathcal{S}$ is *minimal* if $\mathcal{S} \setminus C$ is not a set cover for all $C \in \mathcal{S}$. Minimal set covers have the property that every other cover is a superset of a minimal

[1] `www.guru.com`



(a) Minimum cardinality set cover (b) Minimal set covers

Figure 1: Examples used for illustrating the inadequacy of minimum and minimal set covers in value computations.

cover. In order to estimate the importance of a particular entity C, one could count in how many minimal covers it participates. Although this initially looks like a natural approach, the following example indicates that it can lead to unintuitive results.

EXAMPLE 2. *Consider the instance of the* SET-COVER *problem shown in Figure 1(b). In this instance,* $U = \{u_1, u_2, u_3\}$ *and* $\mathcal{C} = \{C', C, C_1, \ldots, C_m\}$*, and* $C_1 = C_2 = \ldots = C_m = \{u_1\}$*,* $C = \{u_1, u_2, u_3\}$ *and* $C' = \{u_2, u_3\}$*. Observe that* C *can cover all the elements in* U*. Nevertheless, it participates in a single minimal solution* $\{C\}$*. On the other hand,* C' *participates in* m *minimal solutions* $\{C', C_i\}$ *for* $i \in \{1, \ldots, m\}$*. Even though in this case,* C *participates in significantly less minimal solutions than* C'*,* C *is clearly as useful as* C' *(since it is actually a superset of* C'*).*

Counting set covers. We overcome the deficiencies of the above approaches by introducing a score $R(C)$ for every entity $C \in \mathcal{C}$. The *cover score* (or simply *score*) of entity C is defined as follows:

$$R(C) = \sum_{\mathcal{S} \in \mathbf{L}_{\text{sc}}} \delta(C, \mathcal{S})\, w(\mathcal{S}). \quad (1)$$

In the above equation, $\mathbf{L}_{\text{sc}}$ represents *all the set covers* of $\langle U, \mathcal{C} \rangle$; the function $\delta(C, \mathcal{S})$ is an indicator variable that takes value 1 if $\mathcal{S}$ is a set cover and takes value 0 otherwise. The term $w(\mathcal{S})$ assigns to every solution $\mathcal{S}$ a weight such that better solutions – smaller cardinality covers – are assigned higher weights. In other words, for two set covers $\mathcal{S}$ and $\mathcal{S}'$ if $|\mathcal{S}| \leq |\mathcal{S}'|$, then $w(\mathcal{S}) \geq w(\mathcal{S}')$. In other words, important entities participate in many good set covers and therefore are assigned higher scores.

In this paper, we consider three weighting schemes:

- *Uniform:* The uniform scheme assigns a weight of 1 to all set covers. Formally: $w(\mathcal{S}) = 1$ for every $\mathcal{S} \in \mathbf{L}_{\text{sc}}$. For every other $\mathcal{S}$, $w(\mathcal{S}) = 0$.
- *Step:* The step scheme assigns a weight of 1 to all set covers of size less or equal to a threshold τ. Formally: $w(\mathcal{S}) = 1$ if $\mathcal{S} \in \mathbf{L}_{\text{sc}}$ and $|\mathcal{S}| \leq \tau$. Otherwise, $w(\mathcal{S}) = 0$.
- *Cardinality-based:* The cardinality-based scheme assigns a weight that decreases with the cardinality of the set cover. Formally: $w(\mathcal{S}) = w_k > 0$, for every $\mathcal{S} \in \mathbf{L}_{\text{sc}}$ and $|\mathcal{S}| = k$; $w(\mathcal{S}) = 0$ for every $\mathcal{S} \notin \mathbf{L}_{\text{sc}}$.

From the algorithmic point of view, computing $R(C)$ is computationally intractable; note that there are exponentially many solutions in $\mathbf{L}_{\text{sc}}$. More formally, for the uniform weighting scheme, the task of estimating $R(C)$ is identical to the task of counting all set covers for $\langle U, \mathcal{C} \setminus \{C\} \rangle$, which is a #P-complete problem [22].

Discussion: The interpretation of the cover score is the following: if we sample solutions from $\mathbf{L}$ such that every solution $\mathcal{S}$ is sampled with probability proportional to $w(\mathcal{S})$ (i.e., good solutions are sampled with higher probability), then $R(C)$ is proportional to the probability that the sampled solution contains C.

The Equation (1) indicates that the framework we propose bridges Bayesian model-averaging [4] with optimization-based data mining. To draw the analogy, the objective function (expressed via the weight $w(\cdot)$) corresponds to the Bayesian prior. The validity of the solution (expressed via the indicator function $\delta(\cdot, \cdot)$) corresponds to the probability of the data given the model. At a very high level, one can think that $R(C)$ is the average "goodness" of solutions with entity C.

3. ALGORITHMS FOR COUNTING

As already discussed, estimating $R(C)$ is computationally intractable and we cannot hope for computing it exactly in polynomial time. Therefore, we are content with algorithms that approximate the cover score in polynomial time. In this section, we discuss such algorithms for the different weighting schemes.

Apart from $R(C)$ we will also compute the sum of the weights of all the subsets of $\mathcal{C}$. Formally: $R = \sum_{\mathcal{S} \in \mathbf{L}} w(\mathcal{S})$. In some cases, for clarity of presentation, we will use the *uniform weighting scheme*. Under this scheme, R represents the cardinality of set $\mathbf{L}_{\text{sc}}$, namely, the number of set covers for $X = \langle U, \mathcal{C} \rangle$.

3.1 Naïve Monte Carlo sampling

We begin by presenting a naïve Monte-Carlo counting algorithm. We discuss why this algorithm fails to compute $R(\cdot)$ under the uniform weighting scheme, but the discussion applies to the other weighting schemes too.

For input $X = \langle U, \mathcal{C} \rangle$ one could compute estimates $R(C)$ using the following procedure: first, set the counters $r = 0$ and $r(C) = 0$ for every $C \in \mathcal{C}$. Then retrieve N samples from $\mathbf{L}$. For each sample $\mathcal{S}$, check if $\mathcal{S}$ is a set cover. If it is, then: (i) increase the value of r by 1 and (ii) increase the value of $r(C)$ by 1 for all $C \in \mathcal{S}$. In the end of the sampling procedure, report the estimate for R to be $2^m r N^{-1}$ and the estimate for $R(C)$ to be $2^m r(C) N^{-1}$, where m is the number of entities in $\mathcal{C}$.

The above algorithm, which we call `NaiveCounter`, is a standard Monte-Carlo sampling method. Using Chernoff bounds, we can state the following fact [22, Theorem 11.1].

FACT 1. *The* `NaiveCounter` *algorithm yields an* ϵ*-approximation to* $R(C)$ *with probability at least* $1 - \delta$*, provided that*

$$N \geq \frac{4}{\epsilon^2 \rho_c} \ln \frac{2}{\delta},$$

where $\rho_c = \frac{R(C)}{2^m}$.

The drawback of `NaiveCounter` is that, as ρ_c becomes smaller, the number of samples N required to obtain a good approximation increases. Since we do not know the value of ρ_c (in fact, this is what we are trying to approximate) we have to rely on a lower bound. In the absence of any evidence, the only reasonable lower bound is $\rho_c \geq \frac{1}{2^m}$. This, however, requires an exponential number of samples, leading to an inefficient and impractical algorithm.

3.2 Importance sampling

We overcome the problem of `NaiveCounter` with a novel and efficient algorithm, based on the principles of *importance sam-*

Algorithm 1 The `CountSimple` algorithm.

Input: $X = \langle U, \mathcal{C} \rangle$ and seed sets $\mathbf{M} = \{\mathcal{M}_1, \ldots, \mathcal{M}_d\}$.
Output: Quantity $R = \sum_{\mathcal{S} \in \mathbf{L}_{\text{sc}}} w(\mathcal{S})$.

$r \leftarrow 0$
for iterations N, polynomial in d **do**
  pick $\mathcal{M}_\ell$ from $\mathbf{M}$ with prob $\frac{w(\mathbf{M}_\ell)}{\sum_{j=1}^{d} w(\mathbf{M}_j)}$
  $\mathcal{S} = \texttt{RandomSample}(\mathbf{M}_\ell)$
  $\ell^* = \texttt{CanonicalRepresenative}(\mathcal{S})$
  if $\ell^* = \ell$ **then** $r \leftarrow r + w(\mathcal{S}_\ell)$
$r \leftarrow r \frac{w(\mathbf{U})}{N}$
return r

pling.[2] We call our algorithm `Compressed-IC`. Our algorithm works for all our weighting schemes and therefore we present it here in its full generality.

The algorithm requires as input a *seed* of d minimal set covers $\mathbf{M} = \{\mathcal{M}_1, \ldots, \mathcal{M}_d\}$. In principle, *all* minimal set covers are required. However, computing all minimal set covers can be a computationally expensive task. Thus, we propose to use as seed a subset of all the minimal covers. As we show in our experiments this is sufficient to accurately estimate the desired counts. Later in the section, we give details on how to obtain such a seed.

In the sections that follow, we give a bottom-up presentation of the `Compressed-IC` algorithm. That is, we describe a sequence of steps that allow us to modify the classic importance sampling algorithm in order to efficiently approximate the cover scores.

3.2.1 Overview of importance sampling

We start with an overview of importance sampling for the task of estimating R. For a minimal set cover $\mathcal{M}_\ell$ we define $\mathbf{M}_\ell$ to be the collection of all supersets of $\mathcal{M}_\ell$. Given two (small) positive numbers ϵ and δ, the method provides an ϵ-accurate estimate for R with probability at least $(1 - \delta)$, provided that the following three conditions are satisfied:

1. We can compute $w(\mathbf{M}_\ell) = \sum_{\mathcal{S} \in \mathbf{M}_\ell} w(\mathcal{S})$ for all $\ell = 1, \ldots, d$ in polynomial time.
2. We can obtain a random sample from each $\mathcal{S}_\ell$; where the sampling space is determined by the weighting scheme $w(\mathcal{S})$.
3. We can verify in polynomial time if $\mathcal{S} \in \mathbf{M}_\ell$.

For the weighting schemes we consider in this paper, we have the following lemma.

LEMMA 1. *For the uniform, step, and cardinality-based weighting schemes, all three conditions above are satisfied.*

Due to space constraints, we omit the proof of this lemma.

3.2.2 The `CountSimple` algorithm

First, we present an algorithm for estimating $R = w(\mathbf{L})$, the sum of the weights of all the subsets in $\mathbf{L}$, i.e., the collections of all possible subsets of $\mathcal{C}$. Recall that all three considered weighting schemes assign a weight of 0 to all subsets that are not set-covers. Therefore, the problem of computing R is reduced to computing the sum of the weights of all the set covers

$$R = w(\mathbf{L}_{\text{sc}}) = \sum_{\mathcal{S} \in \mathbf{L}_{\text{sc}}} w(\mathcal{S}).$$

The basic idea is to consider the multiset $\mathbf{U} = \mathbf{M}_1 \uplus \ldots \uplus \mathbf{M}_d$, where the elements of $\mathbf{U}$ are pairs of the form $(\mathcal{S}, \ell)$ corresponding to $\mathcal{S} \in \mathbf{M}_\ell$. I.e., for every set cover $\mathcal{S}$, $\mathbf{U}$ contains as many copies of $\mathcal{S}$ as there are $\mathbf{M}_\ell$'s for which $\mathcal{S} \in \mathbf{M}_\ell$. Notice that the total weight of $\mathbf{U}$ can be trivially computed as $w(\mathbf{U}) = \sum_{\ell=1}^{d} w(\mathbf{M}_\ell)$.

The multiset $\mathbf{U}$ is then divided into equivalence classes, where each class contains all pairs $(\mathcal{S}, \ell)$ that correspond to the same set cover $\mathcal{S}$. For each equivalence class, *a single pair* $(\mathcal{S}, \ell)$ is defined to be the *canonical representation* for the class. Now, R can be approximated by generating random elements from $\mathbf{U}$ and estimating the fraction of those that correspond to a canonical representation of an equivalent class. The generation of random samples from $\mathbf{U}$ is done by first picking a minimal set $\mathcal{M}_\ell$ from $\mathbf{M}$ and with probability proportional to the weight of $\mathbf{M}_\ell$. Then, we randomly sample $\mathcal{S}$ from $\mathbf{M}_\ell$. The different weighting schemes imply different sampling methods within $\mathbf{M}_\ell$. However, as Lemma 1 indicates, this sampling can be done in polynomial time.

The pseudocode of this algorithm, named `CountSimple`, is shown in Algorithm 1. The key idea of the algorithm is that instead of sampling from the whole space $\mathbf{L}$, it only draws samples from $\mathbf{U}$. Observe that each cover $\mathcal{S}$ can contribute weight at most $d\, w(\mathcal{S})$ in $w(\mathbf{U})$. Therefore, the ratio $\rho = w(\mathbf{L}_{\text{sc}})/w(\mathbf{U})$ is bounded from below by $1/d$. Following a similar argument as in Fact 1, we can estimate the value of $w(\mathbf{L}_{\text{sc}})$ using $N \geq \frac{4d}{\epsilon^2} \ln \frac{2}{\delta}$ samples, i.e., the number of samples required is polynomial to the size of $\mathbf{M}$.

Algorithm 2 The `ImportanceCounter` algorithm.

Input: $X = \langle U, \mathcal{C} \rangle$ and seed sets $\mathbf{M} = \{\mathcal{M}_1, \ldots, \mathcal{M}_d\}$.
Output: Quantities R and $R(C)$ for every $C \in \mathcal{C}$.

$R \leftarrow 0$
for $C \in \mathcal{C}$ **do**
  $R(C) \leftarrow 0$
for iterations N, polynomial in d, **do**
  pick $\mathcal{M}_\ell$ from $\mathbf{M}$ with prob $\frac{w(\mathbf{M}_\ell)}{\sum_{j=1}^{d} w(\mathbf{M}_j)}$
  $\mathcal{S} = \texttt{RandomSample}(\mathbf{M}_\ell)$
  $\ell^* = \texttt{CanonicalRepresenative}(\mathcal{S})$
  if $\ell^* = \ell$ **then**
    $R \leftarrow r + w(\mathbf{M}_\ell)$
    for $C \in \mathcal{S}_\ell$ **do**
      $R(C) \leftarrow R(C) + w(\mathbf{M}_\ell)$
for $C \in \mathcal{C}$ **do**
  $R(C) \leftarrow R(C) \frac{w(\mathbf{U})}{N}$
$R \leftarrow R \frac{w(\mathbf{U})}{N}$
return R, $R(C)$ for every $C \in \mathcal{C}$

3.2.3 The `ImportanceCounter` algorithm

To estimate the values of $R(C)$, we modify Algorithm 1 as follows: every set $\mathcal{M} \in \mathbf{M}$ needs to be extended so that it contains set C. Using as $\mathbf{M} = \{\mathcal{M}_1 \cup \{C\}, \ldots, \mathcal{M}_d \cup \{C\}\}$, we can then run `CountSimple` for each entity separately and thus compute the number of covers that contain C.

Such an approach, however, is computationally expensive. Each execution of the `CountSimple` algorithm requires N calls to the `CanonicalRepresentative` routine and therefore $O(Nmd)$ time. Since `CountSimple` needs to be executed m times, the total running time is $O(Nm^2d)$. Even if $N = O(d)$ any reasonable seed size is at least $d = O(m)$, implying that the approach needs time $O(m^4)$.

We overcome this computational bottleneck by computing all the score values $R(C)$ for every $C \in \mathcal{C}$ in a single (yet modified) call of the `CountSimple` routine. Algorithm 2 gives the pseudocode of this modified algorithm, which we call `ImportanceCounter`.

[2] See the textbook of Motwani and Raghavan [22] for a detailed discussion on the foundations of this technique.

The main result of this section is that ImportanceCounter yields estimates for the m values of $R(C)$ that are as good as executing m times the CountSimple algorithm, while it requires almost the same number of samples.

THEOREM 1. *If the weighting scheme satisfies a certain "smoothness" condition, then Algorithm 2 gives an ϵ-approximation of $R(C)$ and R with probability $(1-\delta)$ if*

$$N \geq \frac{(1+\alpha)4d}{\epsilon^2} \ln \frac{2}{\delta},$$

where α is a finite integer greater or equal to 1. Furthermore, for the uniform weighting scheme $\alpha = 1$, for the cardinality-based weighting scheme $\alpha = 2$, while for the the threshold weighting there does not exist a finite α for which the above inequality holds.

Although we do not give the proof of the above theorem, we state the smoothness condition that we require for the weighting scheme $w(\cdot)$, so that the above theorem holds for finite α.

CONDITION 1. *If $\mathcal{S}$ and $\mathcal{T}$ are set covers with $|\mathcal{T}| = |\mathcal{S}| + 1$, then the weighting function satisfies $w(\mathcal{S}) \leq \alpha\, w(\mathcal{T})$ for some $\alpha > 0$. In other words, if $\mathcal{T}$ is larger than $\mathcal{S}$ by one element, the value of the weighting function cannot drop too much.*

Running time: Note that the ImportanceCounter algorithm requires only N calls to the GenerateSuperset routine, obtaining a factor of $O(m)$ improvement over the naïve approach of executing CountSimple m times.

3.2.4 *The* Compressed-IC *algorithm*

Even though ImportanceCounter is much more efficient than NaiveCounter, its running time can still be prohibitive for very large datasets. Here, we present the Compressed-IC algorithm, which outputs the same scores as ImportanceCounter, but it is much more efficient in practice.

In many of the datasets we considered, we observed that many of the entities in $\mathcal{C}$ are identical. For example, in the case of reviews, there are many identical reviews that comment on the same product attributes. Similarly, in the case of experts, there are many experts who have exactly the same set of skills. Compressed-IC takes advantage of this phenomenon while optimizing for running time.

One can observe that, for all our weighting schemes, identical entities are assigned the same cover scores. Compressed-IC takes advantage of this observation and compresses all identical entities into into *super-entities*. Then, the algorithm assigns scores to super-entities and in the end it assigns the same score to all the original entities that are part of the same super-entity. The key technical challenge we face here is to design an algorithm that operates on this compressed input and still assigns to all entities the *correct scores*. We say that a score assignment on the compressed input is *correct*, if the scores assigned to entities are identical to the scores that would have been assigned to them by the ImportanceCounter algorithm before compression.

A naïve way of dealing with identical entities in $\mathcal{C}$ is to simply ignore their multiplicities. In this case, all identical entities have a single representative and in the end they are all assigned the score of this representative. Although this procedure will reduce the number of entities and the running time of the ImportanceCounter algorithm, the output scores will be incorrect. In order to see that these scores are incorrect consider the following example.

EXAMPLE 3. *Assume $X = \langle U, \mathcal{C}\rangle$, where $U = \{u_1, u_2, u_3\}$ and $\mathcal{C} = \{C_0, C_1, \ldots, C_K\}$ with $C_0 = \{u_3\}$ and $C_1 = \ldots = C_K = \{u_1, u_2\}$. If we represent all entities $C_1, \ldots, C_K$ by a single representative $\widehat{C} = \{u_1, u_2\}$ we have a new problem instance $X' = \langle U, \widehat{\mathcal{C}}\rangle$, with $\widehat{\mathcal{C}} = \{C_1, \widehat{C}\}$. If we now compute the cover scores of C_0 and $\widehat{C}$ for input X' and for the uniform weighting scheme, we get that both weights are equal to 1. However, in the original problem instance $X = \langle U, \mathcal{C}\rangle$, the score of C_0 is K times larger than the score of any one of the sets $C_1, \ldots, C_K$.*

Instead of assigning the score of the representative to all the identical entities, we could divide this score by the cardinality of each set of identical entities. This solution also fails to produce the correct cover scores. This is shown in the following example.

EXAMPLE 4. *Consider the problem instance $X = \langle U, \mathcal{C}\rangle$, where $U = \{u_1, u_2, u_3, u_4\}$ and $\mathcal{C} = \{C_1, C_2, C_3, C_4\}$ with $C_1 = C_2 = \{u_1, u_2, u_3\}$, $C_3 = \{u_1, u_2\}$ and $C_4 = \{u_3, u_4\}$. Now assume that we form new set $\widehat{\mathcal{C}} = \{\widehat{C}_{12}, C_3, C_4\}$, where $\widehat{C}_{12} = \{u_1, u_2, u_3\}$. For the problem instance $X = \langle X, \widehat{\mathcal{C}}\rangle$ and uniform weighting scheme, the scores assigned to $\widehat{C}_{12}$, C_3 and C_4 are respectively 3, 3, and 4. Now, if we attempt to find the scores in the original problem by dividing the score of $\widehat{C}_{12}$ equally into C_1 and C_2 we get that the final scores are: $R(C_1) = R(C_2) = 1.5$, $R(C_3 = 3)$ and $R(C_4) = 4$. One can see that these scores are incorrect; they do not correspond to the number of solutions that the different entities participate. Even further, ranking the entities using these scores gives a totally unintuitive ranking: C_1 and C_2 appear to be worse than C_3 although they both cover a superset of the elements that C_3 covers!*

The above two examples illustrate that if we want to form one *super-entity* for each set of identical entities in $\mathcal{C}$, we need to make a very meticulous adaptation of the ImportanceCounter algorithm. We describe this adaptation here. First, we transform the input entities $\mathcal{C}$ into $\widehat{\mathcal{C}} = \{\widehat{C}_1, \ldots, \widehat{C}_K\}$ where every $\widehat{C}_j$ is a *super-entity* that represents a set of identical input entities. Each such entity $\widehat{C}_k \in \widehat{\mathcal{C}}$ is characterized by its cardinality T_k. The mechanics of the Compressed-IC algorithm are similar to the mechanics of the ImportanceCounter algorithm, shown in Algorithm 2. The main differences are explained below.

First, note that every minimal set $\mathcal{M}_\ell$ now consists of super-entities and therefore the probability of $\mathcal{M}_\ell$ being sampled is the weighted sum of the probabilities of all the minimal sets it represents. Once a minimal set $\mathcal{M}_\ell$ is selected, a particular element $\mathcal{S}$ is sampled from $\mathbf{M}_\ell$. The sampling is identical to the corresponding step of the ImportanceCounter algorithm. The result of sampling is a set cover $\mathcal{S}$ that consists of j super-entities $\widehat{C}_1, \ldots, \widehat{C}_j$; for each super-entity $\widehat{C}_i$ that participates in $\mathcal{S}$, we know its cardinality T_i as well as the number t_i of actual entities that were sampled from $\widehat{C}_i$. By treating super-entities in the same way as entities, we check whether the canonical representative of the sampled set cover $\mathcal{S}$ is indeed the previously-selected minimal set $\mathcal{M}_\ell$. If this is not the case, no counter is increased. Otherwise, we need to do the following: first, we accept the match with probability $\left(\prod_{\widehat{C}_i \in \mathcal{M}_\ell} t_i\right)^{-1}$. Second, for every entity $C \in \widehat{C}_i$ the score of C is increased by $w(\mathbf{M}_\ell) t_i T_i^{-1}$. The rest of the steps of the Compressed-IC algorithm remain identical with those of the ImportanceCounter algorithm.

We have the following result.

THEOREM 2. *The cover scores computed by* Compressed-IC *are correct. That is, they are identical to the scores computed by the* ImportanceCounter *algorithm.*

Although we omit the proof of Theorem 2 we consider this theorem as the second main technical contribution of our work.

The running time of the Compressed-IC algorithm depends only on the number of super-entities and not on the actual number of entities. Therefore, for datasets that contain small number of distinct entities the Compressed-IC algorithm is extremely efficient, while at the same time it outputs the correct scores for all the individual entities.

From now on, whenever we use the term entities, we will refer to the super-entities that are formed after the compression of the identical input entities.

3.2.5 Selecting the seed of minimal sets

As discussed early in the section, importance sampling requires the complete set $\mathbf{M}$ of all minimal set covers. However, enumerating all the minimal set covers is a computationally expensive task; this problem is known as the *transversal hypergraph* problem. Although we experimented with various available methods for the problem [3, 9, 12], none of them was able to generate all minimal set covers in reasonable time, even for small inputs.

Therefore, we propose to execute Compressed-IC with a seed of minimal sets, which is a subset of $\mathbf{M}$. Our intuition is that even a small-cardinality seed is adequate to accurately estimate the counts of the entities in the input. We form such seeds by starting with set $\mathcal{C}$ and randomly removing elements until we reach a minimal solution. We repeat this process until we generate a seed of the desired size. Our experiments indicate that a seed of size moderately larger than m, where m is the total number of entities, is sufficient.

3.2.6 Decomposition into subproblems

Assume that the input can be partitioned into *disjoint connected components*, where a connected component is defined over the bipartite graph representation that we used in Figure 1. In this case, to compute R and $R(C)$ for every $C \in \mathcal{C}$, it is sufficient to compute the $R(C)$'s for all the C's that are in the same connected component separately. Such separate computations are much faster, since they operate on smaller inputs. We can then obtain the final counts by simply computing the product of the values reported for each component. For the datasets we have experimented with, this decomposition alone has yielded significant speedups. For example, in one of our datasets there were 2227 entities, while the largest component included only 705, i.e., about 30%, of them.

3.2.7 Discussion

Oftentimes, counting problems are solved using the Markov chain Monte Carlo method [20], and Metropolis-Hastings [11]. In our setting, such a method would sample elements of $\mathbf{L}_{\text{sc}}$ by performing a random walk on the space $\mathbf{L}_{\text{sc}}$. Appropriate definitions of the random walk can guarantee unbiased estimates of the required scores. Using the appropriate random walk, we experimented with such a method and found it is very inefficient when compared to Compressed-IC. Thus, we do not present this method in detail, neither do we show any experiments for it.

4. EXPERIMENTS

In this section, we describe the experiments we conducted to evaluate the Compressed-IC algorithm, in terms of both utility and performance. We also report the results of a user study that demonstrates the validity of our scores and their closeness to the human perception of important entities. All our results are obtained using the step weighting scheme, since it is the obvious choice in applications where the size of the set cover solutions is important. For all our experiments, we use a machine with 8GB of RAM and 2.66GHz processor speed.

4.1 Datasets

We evaluate our algorithms using three families of datasets, which we describe below.

GPS **datasets:** We consider the reviews of the five most-frequently reviewed GPS systems sold on Amazon.com, as of 2010. Thus we compile five different datasets in total. For this dataset family, the universe U is the set of product attributes. Each review C_i comments on a subset of the product attributes. In other words, each review is represented by the set of the attributes discussed in the review. We construct the subsets C_i by parsing the reviews using the method by Hu and Liu [13], which can be shown to extract the review attributes with high accuracy. An attribute is covered if there exists at least one review that discusses this attribute. The cover score $R(C_i)$ expresses the usefulness of the review C_i. In addition to the complete text for each review, the datasets also include the number of the accumulated *helpfulness* votes of each review; we anonymize the five datasets and simply refer to them as GPS-1, GPS-2, GPS-3, GPS-4 and GPS-5. The number of reviews in these datasets are 375, 285, 254, 216 and 222, respectively.

Guru **datasets:** We construct the Guru datasets by crawling the www.guru.com website, which hosts profiles of active freelancers from various professional disciplines, such as software engineering, marketing, etc. We extracted the skills of each each freelancer and we compiled five different datasets as follows: each dataset corresponds to a task that requires c skills, with $c \in \{10, 20, 40, 80, 160\}$. Each dataset consists of all the freelancers who have at least one of the skills required by the respective task. We refer to the five datasets as Guru-1, Guru-2, Guru-3, Guru-4 and Guru-5. The number of freelancers in each one of these datasets is 109 158, 184 520, 239 856, 300 256 and 317 059 respectively.

With the Guru datasets, we evaluate the ability of our algorithm to identify the freelancers who are most valuable for a given task. Consequently, the universe U is the set of skills required for the task. Each freelancer C_i is represented by the set of his skills, as they appear in his profile. The cover score $R(C_i)$ given by our algorithm quantifies the usefulness of the freelancer C_i, with respect to the given task and can be used to rank experts.

4.2 Performance of Compressed-IC

In this section we provide a comprehensive experimental evaluation of the behavior of the scores $R(C)$ output by Compressed-IC as well as the running time of the algorithm.

Inefficiency of the naïve Monte-Carlo sampling: We start the presentation of our experimental results by pointing out that the the naïve Monte-Carlo sampling technique presented in Section 3.1 is impractical. The reason for that is that the naïve sampling rarely succeeds in forming a set from $\mathbf{L}_{\text{sc}}$. In practice, for each one of our 10 datasets, we sampled 10 000 sets from $\mathbf{L}$: in 5 out of 10 datasets 0% of the samples were set covers; in 4 out of 10 1% of the samples were set covers and in the remaining one 12% of the samples were set covers. This indicates, that we cannot rely on this technique in order to compute the $R(C)$ scores.

Compressed-IC **vs.** ImportanceCounter**:** As we discussed in Section 3.2.4, the Compressed-IC algorithm gives the same results as ImportanceCounter, but it is designed to handle datasets with multiple copies of entities in $\mathcal{C}$. Thus, the only difference between the Compressed-IC algorithm and ImportanceCounter is that the former keeps a single copy for every set of identical entities, rather than considering each one of them separate-

Table 1: Compressed-IC; GPS and Guru datasets: ratio of the number of super entities considered by Compressed-IC over the number of original set of entities in the dataset.

GPS				
GPS-1	GPS-2	GPS-3	GPS-4	GPS-5
0.808	0.6667	0.7087	0.8009	0.6667
Guru				
Guru-1	Guru-2	Guru-3	Guru-4	Guru-5
0.0001	0.0004	0.0005	0.0036	0.0253

ly. In other words, the Compressed-IC algorithm compresses entities into super-entities. Table 1 shows the corresponding *compression ratio*; that is, the ratio of the number of super-entities (after compression) to the number of total entities (before compression) in all 10 datasets. The smaller the value of this ratio the larger the compression achieved by the super-entities.

The results show that the compression ratio achieved for the GPS datasets ranges from 67% to 81%. For the Guru datasets, this ratio is much smaller, ranging from 0.01% to 2.5%. Such large compressibility can be explained by the nature of this particular data type; recall that, in the Guru datasets, each freelancer is associated with a set of skills. It is much more likely to have multiple freelancers with identical sets of skills; for example, it is expected that all programmers know more or less the same set of programming languages. On the other hand, when it comes to product reviews, it is less likely (although possible) to find reviews that comment on exactly the same set of product attributes.

Effect of the size of the seed of minimal sets: As we discussed in Section 3.2, ImportanceCounter requires as input a set of minimal solutions. While, in principle, the complete set of all such solutions is required, we show here that using a subset is sufficient for the algorithm to converge. First, we run the algorithm with different seed sizes and retrieve the produced cover scores for each individual. Specifically, we try all the seed sizes in $\{m, 10m, 20m, 30m, 40m\}$ (in that order), where m is the total number of entities. We then compare the scores obtained for the different entities after consecutive runs (i.e., m vs. $10m$, $10m$ vs. $20m$, etc.). Since we use these scores to rank entities, we assess the stability of the different runs by comparing the ranked lists induced by the computed scores. For comparing ranked lists, we use the popular Kendall-τ rank correlation coefficient [15]. The measure takes values in $[0, 1]$; the lower the value the more different the two lists are.

The process is repeated for all 10 datasets. The results are shown in Table 2. Each value is an average of 50 experiments, each using a different seed set. The results show that even though we use very small seed sets, the scores obtained with seeds of different sizes produce very similar ranked lists. In particular, as the size of the seed set grows very moderately most values of τ become very close to 1.00. This behavior is verified across all three families of datasets. Therefore, we conclude that the Compressed-IC algorithm is robust, and the estimated scores produce stable rankings even when considering small-size seed sets.

Effect of the number of samples: Theorem 1 provides a theoretical bound on how many samples are necessary for the ImportanceCounter and Compressed-IC algorithms compute the true values of $R(C)$ under different weighting schemes. In this experiment, we demonstrate that, in practice, even smaller number of sample is sufficient.

Table 2: Kendall-τ rank correlation between rankings induced from cover scores obtained with seeds of different sizes. The results are averages of 50 repetitions, where each repetition uses a different seed.

Dataset	1 vs. 10	10 vs. 20	20 vs. 30	30 vs. 40
GPS-1	0.75	0.77	0.82	0.86
GPS-2	0.77	0.88	0.90	0.90
GPS-3	0.82	0.88	0.9	0.9
GPS-4	0.8	0.9	0.9	0.9
GPS-5	0.79	0.85	0.87	0.89
Guru-1	1.00	1.00	1.00	1.00
Guru-2	0.88	0.93	0.93	0.94
Guru-3	0.84	0.89	0.89	0.90
Guru-4	0.96	0.99	0.99	0.99
Guru-5	0.98	0.98	0.99	0.99

Table 3: Number of samples and time required until the Kendall-τ rank correlation coefficient reaches a value 0.90. The results are averages of 50 repetitions, with 50 different seeds.

Dataset	# entities	# samples	time (sec)
GPS-1	375	3 000	4.222
GPS-2	285	3 000	1.526
GPS-3	254	3 000	1.14
GPS-4	216	3 000	1.130
GPS-5	222	3 000	0.689
Guru-1	109 158	8 640	29.607
Guru-2	184 520	6 600	42.547
Guru-3	239 856	3 920	26.595
Guru-4	300 256	3 100	71.367
Guru-5	317 059	3 060	47.552

Our experimental setting is the following: first we apply the Compressed-IC algorithm to each of the 10 datasets, using a seed of size $20m$. As a convergence criterion, we use the Kendall-τ rank correlation coefficient. We let the algorithm run until the Kendall-τ coefficient reaches a value of 0.90 (while checking the τ coefficient every 1000 samples).

The results of our experiments are summarized in Table 3. Again, the results reported in Table 3 are averages over 50 repetitions, where each repetition considers a different seed set. For each dataset, in the second column of the table, we report the number of entities in the dataset since it has direct impact on the complexity of algorithm.Next, we report the total number of total samples required until reaching Kendall-τ value of 0.90 (third column), as well as the actual computation time in seconds (fourth column).

As shown in Table 3, in most cases a small number of samples is needed: in all cases but one the number of samples required is less than 10 000. This is true even for the Guru datasets, which have three orders of magnitude more sets than the other datasets. Therefore, the number of required samples grows very mildly with the number of sets in the dataset. On the other hand, the number of entities impacts the overall running time of the algorithm. This is expected since the time required for drawing one sample grows linearly with the number of entities in the dataset.

4.3 Evaluating the quality of the results

Here, we compare the scores produced by Compressed-IC with other baselines. Due to space constraints, we only show results for the largest dataset from each family.

For the GPS-1 dataset, we use as baseline score the *helpfulness score* of a review, i.e., the number of users that have declared this review to be helpful. The helpfulness score is a popular criterion

for ranking reviews in review-hosting sites (e.g., amazon.com). Figure 2(a) shows the scatterplot between the cover and the helpfulness scores; note that the cover scores are normalized, i.e, scores $R(C)$ are divided by R so that the measures are in $[0, 1]$.

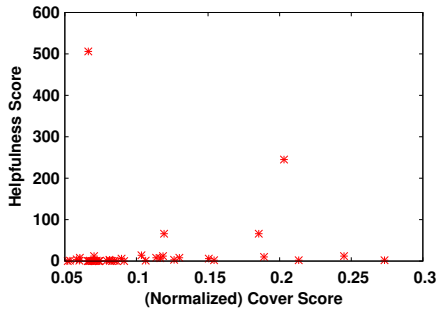
We observe that there is no correlation between the cover and helpfulness. For example, helpfulness does not directly consider the contribution of each review in the coverage of an item's attributes. Instead, it relies on the willingness of the users to reward a "good" review by giving a helpfulness vote. In such a scheme, older reviews are more likely to collect higher helpfulness votes, regardless of whether they are more informative [19]. Our scoring scheme, on the other hand, is unaffected by such biases, since it objectively evaluates reviews based on their contribution.

For the Guru-5 dataset, we select a *frequency-based* baseline score, i.e., the number of skills of each freelancer. The scatter plot between our measure and the baseline is shown in Figure 2(b). This baseline measure can take only one of 5 distinct values {1,2,3,4,5} and considers all freelancers with the same number of skills as equally important. This is an over-simplifying assumption that does not take into consideration factors like the rarity of each skill. For example, one of the freelancers has the following five skills: {*Children's Writing, Writing-Editing & Translation, Reports, Technical Writing, Blogs*}. While this freelancer has the maximum number of skills, all of these skills are very common. This fact is captured by our own measure, since the particular freelancer participates in far less covers than most of his peers. As a result, this freelancer is given one of the lowest observed scores (0.007). On the other hand, a freelancer with only three skill {*Commercial Interior Design*, *Landscape Design*} is assigned one of the top-30 scores (0.49). This is because his second skill (*Landscape Design*) is a rather rare skill, possessed by only 260 other individuals. Therefore, our scoring scheme can differentiate between individuals with the same number of skills, and it can also identify important experts even when they have small number of skills.

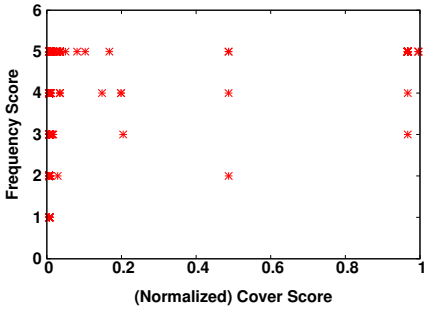
4.4 User study

In this section, we present a user study that we conducted in order to validate the assumption that cover scores give rankings that agree with human intuition. Using the original dataset from Guru, we sampled 10 small groups of freelancers; the cardinality of each group was randomly selected between 5 and 10. For the i-th group of freelancers we created a task T_i, which was defined as the union of the skills of the freelancers in the i-th group. Using these 10 tasks we set up a survey on kwiksurveys.com, in which 30 human subjects participated. The subjects were asked to inspect the skills required for a task and the skills of the freelancers in the corresponding group. The subjects were then asked to rank the freelancers in the group by assigning to every freelancer a numeric "usefulness" score with the following interpretation: 4 if the freelancer is absolutely necessary, 3 if she is very useful, 2 if she is somewhat useful, and 1 if she is not useful. In the instructions it was made clear that the team will have to complete the task repetitively and that the task can only be completed if all the required skills are represented.

Figure 3 shows the average Kendall-τ rank correlation coefficient between the rankings obtained by the humans and the rankings obtained by three automated measures: (i) using the cover scores, (ii) the number of skills per freelancer, and (iii) the minimum set-cover solution; in the latter ranking all freelancers in the minimum set cover were ranked higher than the rest. Recall that the Kendall-τ coefficient takes values in $[0, 1]$, with 1 (resp. 0) denoting the largest (resp. smallest) similarity between the compared rankings. The results indicate that the ranking obtained by cover scores has consistently high Kendall-τ value; this value never drops below 0.8 and it is close to 1 for most of the tasks. On the other hand, with the exception of task T_4, the Kendall-τ coefficients between human rankings and rankings by the number of skills per freelancer are notably low. For task T_4, we observe the highest similarity between these two rankings. This is because the freelancers in the 4-th group are more or less equivalent both in terms of their count scores and in terms of their number of skills (all of them have 1 or 2 common skills). Therefore, the frequency and the cover-score rankings are very similar to each other and also similar to the human rankings. This also explains why for task T_4 the minimum set-cover ranking does not correlate well with the human ranking. Apparently, the single set-cover solution does not reflect the equivalence between the freelancers and the fact that they all participate in many different set-cover solutions. Other than T_4, the minimum set-cover rankings correlate well with the human rankings. This is mostly because for many of the tasks we considered the number of freelancers is small and therefore the majority of the freelancers



(a) GPS-1 dataset



(b) Guru-5 dataset

Figure 2: Scatterplots of (normalized) scores of entities against their baseline scores.

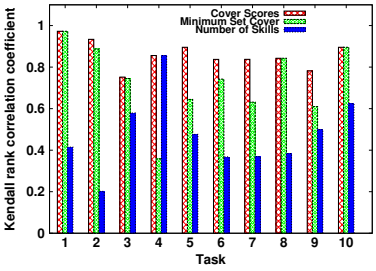


Figure 3: Average Kendall-τ rank-correlation coefficient between human rankings and rankings obtained by cover scores, number of skills and minimum set covers.

participate in the minimum cover. Nevertheless, this correlation is not as strong as the one between human and cover-score rankings.

Overall, the user study demonstrates that humans perceive cover scores as a natural ranking mechanism.

5. RELATED WORK

To the best of our knowledge, we are the to formalize framework that allows counting techniques for the evaluation of entity importance. Nonetheless, our work has ties to existing research. We summarize some of this work here.

Counting set covers. There are many theoretical studies on the problem of hitting-set counting, [6, 7, 8, 10] including complexity studies and algorithms for special inputs. Since the hitting set and the set cover problems are isomorphic, these methods can also count set covers. Among these works, the closest to ours is the work of Damaschke and Molokov [8], that proposes a parameterized algorithm for counting all k-hitting sets (i.e, hitting sets of size at most k) in set systems, where the size of the maximum set is at most r. Their algorithm runs in time that is exponential in k and in r, and despite its theoretical elegance, it is impractical for reasonably large datasets.

Review selection. Lappas and Gunopulos [16] considered the problem of finding a small set of reviews that cover *all* product attributes. Tsaparas et. al. [23] studied the problem of selecting a set of reviews that includes both a positive and negative opinion on each attribute. More recently, Lappas and Terzi [18] proposed a scheme for showing to users different covers of the same corpus of reviews. All these works focus on picking a single set of reviews that optimizes a particular function under a set of constraints. On the other hand, our framework evaluates reviews by assigning a score based on the weight of *all* the solutions that each review participates in. Although the notion of coverage is central to our framework, our intuition and methodology are distinct from existing work in the area.

Team formation. The problem of identifying a set of individuals from a pool of experts who are capable of performing a given task has been an active area of research in Operations Research [5, 25, 26]. Our own recent work has introduced the problem in the computer science domain [2, 17]. This work focuses on identifying a single team of experts that collectively meet a certain set of requirements. Our framework is complementary to this: it identifies the marginal value of each expert with respect to a given task. Also, instead of reporting a single solution of the optimization problem at hand, our framework counts *all* solutions. Thus, the technical challenges we face are orthogonal to the those faced when looking for a single good solution.

6. DISCUSSION AND CONCLUSIONS

Our work is motivated by data-mining problems that have been formalized in terms of the set-cover problem. For such formulations, we have developed a novel framework for evaluating the importance of entities. Instead of looking at a single set-cover solution, our framework computes the importance of entities by counting of the number of good set covers an entity participates. Our algorithmic contribution is the design of `Compressed-IC`, which is an efficient algorithm for solving this problem. Our algorithm is proven to provide the correct counts and scales extremely well. Our framework has applications in numerous domains, including those of human-resource management and review-management systems. In a thorough experimental evaluation, we have demonstrated the efficiency and the effectiveness of our methods, using real datasets from such domains.

Acknowledgements. Aristides Gionis was partially supported by the Torres Quevedo Program of the Spanish Ministry of Science and Innovation, co-funded by the European Social Fund, and by the Spanish Centre for the Development of Industrial Technology under the CENIT program, project CEN-20101037, "Social Media" (`http://www.cenitsocialmedia.es/`).

Theodoros Lappas and Evimaria Terzi were supported by the NSF award #1017529, and from gifts from Microsoft, Yahoo! and Google.

7. REFERENCES

[1] F. N. Afrati, A. Gionis, and H. Mannila. Approximating a collection of frequent sets. In *KDD*, 2004.

[2] A. Anagnostopoulos, L. Becchetti, C. Castillo, A. Gionis, and S. Leonardi. Power in unity: forming teams in large-scale community systems. In *CIKM*, 2010.

[3] J. Bailey, T. Manoukian, and K. Ramamohanarao. A fast algorithm for computing hypergraph transversals and its application in mining emerging patterns. In *ICDM*, 2003.

[4] C. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

[5] S.-J. Chen and L. Lin. Modeling team member characteristics for the formation of a multifunctional team in concurrent engineering. *IEEE Transactions on Engineering Management*, 51(2), 2004.

[6] V. Dahllöf, P. Jonsson, and M. Wahlström. Counting models for 2sat and 3sat formulae. *TCS*, 332(1-3), 2005.

[7] P. Damaschke. Parameterized enumeration, transversals, and imperfect phylogeny reconstruction. *TCS*, 351, 2006.

[8] P. Damaschke and L. Molokov. The union of minimal hitting sets: Parameterized combinatorial bounds and counting. *J. Discrete Algorithms*, 7(4), 2009.

[9] T. Eiter and G. Gottlob. Identifying the minimal transversals of a hypergraph and related problems. *SIAM J. Comput.*, 24, 1995.

[10] J. Flum and M. Grohe. The parameterized complexity of counting problems. *SIAM J. of Computing*, 33(4), 2004.

[11] W. K. Hastings. Monte Carlo sampling methods using Markov chains and their applications. *Biometrika*, 57(1), 1970.

[12] C. Hébert, A. Bretto, and B. Crémilleux. A data mining formalization to improve hypergraph minimal transversal computation. *Fundam. Inf.*, 80, 2007.

[13] M. Hu and B. Liu. Mining opinion features in customer reviews. In *AAAI*, 2004.

[14] D. Kempe, J. M. Kleinberg, and É. Tardos. Maximizing the spread of influence through a social network. In *KDD*, 2003.

[15] M. G. Kendall. A New Measure of Rank Correlation. *Biometrika*, 30(1/2), 1938.

[16] T. Lappas and D. Gunopulos. Efficient confident search in large review corpora. In *ECMLPKDD*, 2010.

[17] T. Lappas, K. Liu, and E. Terzi. Finding a team of experts in social networks. In *KDD*, 2009.

[18] T. Lappas and E. Terzi. Toward a fair review-management system. In *ECMLPKDD*, 2011.

[19] J. Liu, Y. Cao, C. Y. Lin, Y. Huang, and M. Zhou. Low-Quality Product Review Detection in Opinion Summarization. In *EMNLP-CoNLL*, 2007.

[20] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller. Equation of State Calculations by Fast Computing Machines. *The Journal of Chemical Physics*, 21(6), 1953.

[21] T. Mielikäinen and H. Mannila. The pattern ordering problem. In *PKDD*, 2003.

[22] R. Motwani and P. Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995.

[23] P. Tsaparas, A. Ntoulas, and E. Terzi. Selecting a comprehensive set of reviews. In *KDD*, 2011.

[24] V. Vazirani. *Approximation Algorithms*. Springer, 2003.

[25] H. Wi, S. Oh, J. Mun, and M. Jung. A team formation model based on knowledge and collaboration. *Expert Syst. Appl.*, 36(5), 2009.

[26] A. Zzkarian and A. Kusiak. Forming teams: an analytical approach. *IIE Transactions*, 31, 2004.