Discovering Enterprise Concepts Using Spreadsheet Tables

Keqian Li*
Univ. of California, Santa Barbara
klee@cs.ucsb.edu

Yeye He
Microsoft Research
yeyehe@microsoft.com

Kris Ganjam
Microsoft Research
krisgan@microsoft.com

ABSTRACT

Existing work on knowledge discovery focuses on using natural language techniques to extract entities and relationships from textual documents. However, today relational tables are abundant in quantities, and are often well-structured with coherent data values. So far these rich relational tables have been largely overlooked for the purpose of knowledge discovery. In this work, we study the problem of building concept hierarchies using a large corpus of enterprise spreadsheet tables. Our method first groups distinct values from tables into a large hierarchical tree based on co-occurrence statistics. We then "summarize" the large tree by selecting important tree nodes that are likely good concepts based on how well they "describe" the original corpus. The result is a small concept hierarchy that is easy for humans to understand and curate. Our end-to-end algorithms are designed to run on Map-Reduce and to scale to large corpus. Experiments using real enterprise spreadsheet corpus show that proposed approach can generate concepts with high quality.

ACM Reference format:
Keqian Li, Yeye He, and Kris Ganjam. 2017. Discovering Enterprise Concepts Using Spreadsheet Tables . In *Proceedings of ACM conference on Knowledge Discovery and Data Mining, Nova Scotia, Canada, Aug 2017 (KDD'17),* 10 pages.
DOI: http://dx.doi.org/10.1145/3097983.3098102

1 INTRODUCTION

Existing research on knowledge discovery [2, 4, 10, 21, 30, 43, 49] heavily relies on text documents and natural language techniques. This has led to a long and fruitful line of research, and resulted in influential systems and knowledge bases such as NELL [10], KnowItAll [21], TextRunner [4], Probase [49], etc. However, our observation is that so far the success has been limited to the public *web domain*. In addition to this one single web domain that has been heavily studied, there also exist numerous *enterprise domains*, each of which has its own entities and concepts that share little overlap with others (e.g., each enterprise will have its own product categorizations, cost-center classifications, and organizational hierarchies, etc.). Like knowledge bases from the web, such structured knowledge from enterprise domains has the potential to bring significant benefits to a large class of enterprise applications like keyword search [23, 33, 44], data integration [3, 17], etc. However, knowledge discovery from enterprise data is a largely unexplored topic to this date.

Concept	Example Entities
Profit Center	P73126 US - DPE Central Mgmt, P19059 US-HealthCare Provider, . . .
Service Office	US-EDU-EAST, US-PubSec-HQ, US-DOD-ARMY, US-CSUS, . . .
Data Center	34728-Canyon Park, 40112-San Antonio IDC5 40331-Amsterdam, 40590-Tukwila 2, . . .
Windows Protocol	MS-WSUSOD, MS-NETOD, MS-AUTHSOD MS-CAPR, MS-ISTD, MS-RDPEI, . . .

Table 1: Example concepts and entities from an enterprise spreadsheet corpus (some values are redacted for anonymization).

Despite the success of natural language based techniques in the web domain, our experience suggests that they do not work well when applied to enterprise domains for two main reasons. First, text documents are scarce in enterprises when compared to the general web. On the web, the abundance of documents ensures that common sense knowledge are repeatedly embodied in text patterns (e.g. the pattern "`countries such as USA, Canada, ...`" may be mentioned thousands of times). In comparison, in enterprises such text patterns are less frequent because the amount of text data is often limited. Using Microsoft's intranet search engine, we only found a few hits for the text pattern "`Microsoft products such as Microsoft Office 2016`". Replacing "`Office 2016`" with less popular names such as "`SQL Server 2016`" would yield no hits. We estimate text data in enterprises to be orders of magnitude less in quantity than the general web, thus creating difficulty for text-pattern based approaches that heavily rely on data redundancy.

The second drawback of natural language based techniques is that they tend to generate instances that are not entirely compatible with each other [29]. For instance, in addition to extracting "`Microsoft Office 2016`" from the sentence above, such techniques would also extract "`Office 2016 professional plus`" (from a sentence "`Microsoft products such as Office 2016 professional plus`"), as well as other mentions like "`Office 2016`", "`Office`", "`Excel 2016`", etc., which are clearly at different conceptual levels. These entities are not entirely compatible with each other to be lumped into the same concept, and can require substantial post-processing efforts to derive clean concept-entity relationships. Unlike the web data where the cleaning efforts on one general-purpose knowledge base can be amortized, every enterprise has its own proprietary data and would require separate cleaning efforts, making it impractically expensive to scale to a large number of enterprises.

Our observation is that tabular relational data are ubiquitous in enterprises in the forms of spreadsheet files (.xls files and other related formats). In Microsoft intranet for example, we crawled

*Part of work done at Microsoft.

KDD'17, Nova Scotia, Canada

DOI: http://dx.doi.org/10.1145/3097983.3098102

and extracted over 500K tables from enterprise spreadsheets [16], each of which typically has hundreds of rows and tens of columns, covering a wide variety of topics. Furthermore, we observe that data values in these tables are often clean and well-structured, with coherent sets of related entities in same columns (e.g., list of products, list of cost center names, etc.), where it is actually relatively uncommon for incompatible entities like "`Office`", "`Office 2016`", "`Excel 2016`" and "`Office 2016 professional plus`" to be all mixed in the same columns. Table 1 gives a few more examples of enterprise-specific concepts in spreadsheet corpus.

The abundance of relational tables and their structured nature make them suitable for knowledge discovery in enterprises. But existing techniques mostly rely on natural language texts [2, 10, 30, 42, 43, 49] and are not directly applicable. Relational web tables are used in the limited context of enriching preexisting knowledge bases in the web domain [6, 19, 35, 50], by essentially leveraging entities already in knowledge bases as seeds. Such techniques are unfortunately inapplicable to enterprise settings, since no knowledge bases exist in enterprises that these techniques can bootstrap from.

In this work, we take the first steps toward using relational tables for concept discovery in enterprise domains. Specifically, we hope to discover coherent sets of entities that collectively form concepts. It is worth noting that while HTML web tables have been used for *set expansion* [1, 27, 38, 46], these techniques work in an *online query* fashion, in which a concept name and/or seed entities are provided as input, in order to discover additional entities. In comparison, we in this work attempt to discover many concepts simultaneously offline from a large table corpus, without the need of seed entities and concept names as input.

The challenge of using tables for concept discovery is that while table columns are relatively structured, they can be arbitrary *subsets* and *supersets* of ground truth concepts. For example, entities in a column may be *incomplete* with only a subset of entities for that concept. Similarly, some columns may contain *irrelevant* values, either by mixing related concepts (e.g., cities and countries in an "address" column), or mixing meta-data with data (e.g., column headers, section headers, and tally entries like "sum" and "total").

Our main approach is to leverage co-occurrence statistics of values from a large spreadsheet table corpus. While individual table columns may have irrelevant values mixed in, these outlier values can be identified as having low statistical correlation with true members of that concept (e.g. cities vs. countries, or cities vs. "total"), because the true members should occur repeatedly and more often than others. Using that observation, we first construct an exhaustive clustering tree for all values, and then select nodes that are likely to correspond to important concept. The resulting concept hierarchies can be manually inspected and curated by domain experts to ensure high accuracy, which can then be used to benefit an array of enterprise applications.

Contribution Our work makes the following contributions.

- We studied the novel problem of automatic concept discovery from spreadsheet tables, without using concept names or seed instances as input.
- We proposed a scalable and data-driven approach to discover concept hierarchies on Map-Reduce. Our results are easy for humans to understand and amenable to curation.
- Our evaluation using real enterprise spreadsheet corpus, and human labeled benchmarks suggests that the proposed approach can discover concepts with high quality.

2 SOLUTION OVERVIEW

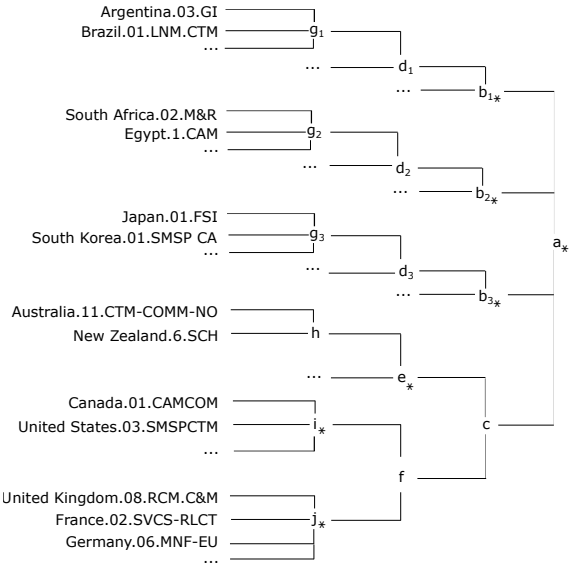
At a high level, our end to end problem can be stated as follows: given an input table corpus represented as a set of *table columns*, denoted as $\mathbf{C}$. Let $\mathbf{V}$ be the universe of all values present in the corpus, each column $c \in \mathbf{C}$ consists of a set of cell values, or $c \subseteq \mathbf{V}$. Our goal of concept discovery is to produce a hierarchical tree, denoted as (O, E), where $O = \{o | o \subseteq \mathbf{V}\}$ are nodes in the tree, each of which consists of a set of values in $\mathbf{V}$ and ideally corresponds to a real-world concept. The edges of the tree in $E \subset O \times O$ represent parent-child relationships between nodes in O, which are determined based on containment of values. Note that in this work we only produce clusters of values that should belong to the same concept, and do not attempt to generate textual "concept names" for them. Producing concept names from entities is an interesting but generally orthogonal topic [16].

At a high level, our approach has two main steps. In the first step, we compute statistical co-occurrence scores for all pairs of values $v_1, v_2 \in \mathbf{V}$, to determine their strength of semantic relationship[1]. We then iteratively merge related values in a bottom-up, hierarchical manner. This is natural because concepts often follow a hierarchical tree structure [32], which starts as fine-grained concepts at leaf levels, and gradually generalizes into broader concepts as one moves up in the tree. In this step we perform exhaustive merging based on co-occurrence, resulting in a deep tree of clusters. Note that many intermediate nodes in the deep tree may not correspond to useful concepts, and the large amount of results makes it difficult for humans to consume directly. This is by design for the first step – we want to preserve as many candidates as possible here, and leave the important decisions of selecting useful concepts to the next stage. We call the resulting candidate tree a *deep clustering tree*, denoted as (O_c, E_c), where $O_c = \{o | o \subseteq \mathbf{V}\}$, and $E_c \subset O \times O$, which becomes input for the next step.

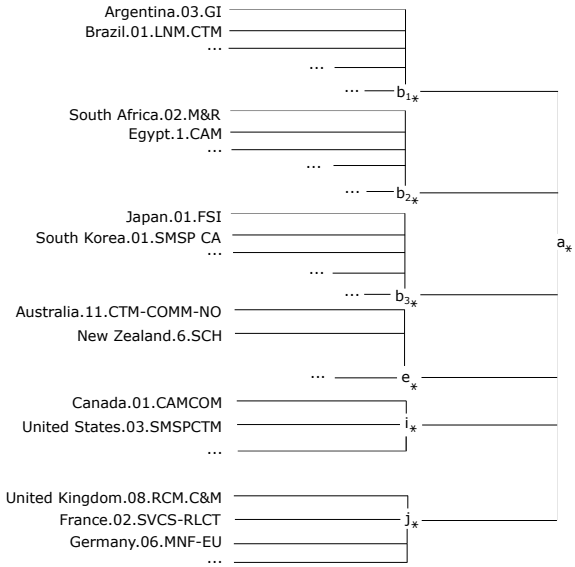
This first step is conceptually straightforward as it performs bottom-up clustering, but the challenge we face is to scale to a large graph with over 10M entities (the size of $|\mathbf{V}|$). For this we propose to use batch clustering and adapt the random-mate algorithm [36] to this setting. We mostly use known techniques for this pre-processing stage.

In the second step, which is the focus of this work, we take the deep clustering tree (O_c, E_c) as input, and judiciously select a subset of nodes $O \subseteq O_c$ that likely correspond to good concepts as output. The selected O induces a new reduced tree (O, E) from (O_c, E_c), where the high-level goal is to preserve as many useful concepts as possible, while making the reduced tree amenable to human understanding and curation. We formulate this tree-reduction step as an optimization problem, where the objective is to find an optimal set O, whose quality is measured by how well O can "describe"

[1]Note that using techniques from the all-pair-similarity literature (e.g. [5]), we do not need to compute the full Cartesian product since most pairs of values have zero co-occurrence and can be ignored.



(a) Step-1: A deep tree by bottom-up clustering. Nodes in the tree are candidates for reduction in the next step.



(b) Step-2: Tree reduction with height=2, by selecting all nodes marked with * from the previous step.

Figure 1: Illustration of the two steps in our approach using an enterprise concept called ATU (account team unit) specific to this spreadsheet corpus. Some values are slightly redacted for anonymization.

the original corpus (to be defined formally), with the constraint that the height of reduced-tree (O, E) to be no more than some small constant h (e.g., 2 or 3). One way to look at this is that we are effectively compressing the deep clustering tree into a smaller tree, using rationales consistent with well-known principles such as *minimum description length* [37] and *Occam's razor* [24]. We develop dynamic programming algorithms on Map-Reduce to solve this problem optimally, which can also scale to large corpus.

We illustrate these two steps of our approach using a simplified example in Figure 1.

Example 1 (Solution Overview). *Figure 1 shows a set of codes names from enterprise spreadsheets that denote a concept called ATU (account team unit), which is an enterprise-specific concept in this corpus that divides teams of customer relationship specialists into subgroups, based on their geographical locations as well as functions. As we can see from the figure, instances in this concept are encoded with a geographical component, followed by a subdivision number, and finally some abbreviated names.*

In the first stage of our algorithm as shown in Figure 1a, we iteratively group values in fine-grained steps. Entities with high co-occurrence will merge first to form lower-level tree nodes. They are iteratively merged with other co-occurring values in a bottom up fashion, forming broader concepts. This terminates when reaching the root level, producing a deep clustering tree which is the result of our first step.

We can see from this particular tree that individual ATUs generally group according to geographical-affinity, reflecting their strength of occurrence in the large spreadsheet corpus. For example, ATUs from the same countries and continents tend to group together first, because many of the columns in the corpus contain exclusively ATUs in the same country, or ATUs with the same continent.

Note that the result of Figure 1a is a deep clustering tree with a large number of intermediate nodes (denoted by different letters), many of which may not correspond to useful concepts, and the tree is likely too big for users to consume. The second stage of our approach produces the result shown in Figure 1b, which in this case is a shallow tree of height 2. The constraint on height h is a parameter that users can provide, which controls the complexity of the final result (a tree of smaller height is generally easier for humans to understand, at the risk of losing certain concepts). This step is mainly based on how well a selected subset of nodes can "summarize" the original corpus, and will result in a shallow tree of concepts, which is easy for humans to understand/curate.

By first exhaustively generating a deep tree with fine-grained value groupings as candidates, and then summarizing the tree, we capture high quality concepts while limiting the size of the output.

3 CANDIDATE TREE GENERATION

In this section, we describe the first step of our approach, which produces a deep clustering tree by exhaustive grouping of values in a spreadsheet corpus, based on their strength of co-occurrence. For this step, while scaling the bottom-up clustering to a large corpus is non-trivial, the conceptual task is relatively straightforward and not the focus of this work, so we briefly discuss this step for completeness.

Given a large table corpus, we use statistical value co-occurrence between any pair of values to capture their semantic relatedness in a data-driven manner. For example, in Figure 1, if a value (e.g., "`France.02.SVCS-RLCT`") co-occurs frequently with another (e.g.,

"Germany.06.MNF-EU") in the same columns, then intuitively they will have a high relatedness score. We define relatedness score $s(v_1, v_2) \in [0, 1]$ using Jaccard similarity as follows.

Definition 1 (Value Similarity by Jaccard). *The relatedness score between two table value v_1, v_2 using Jaccard similarity can be defined as* $s(v_1, v_2) = \frac{|\{c|v_1 \in c, c \in \mathbf{C}\} \cap \{c|v_2 \in c, c \in \mathbf{C}\}|}{|\{c|v_1 \in c, c \in \mathbf{C}\} \cup \{c|v_2 \in c, c \in \mathbf{C}\}|}$.

In principle, any set-based, vector-based and distribution-based similarity metric such as Point-wise Mutual Information, Dice coefficient, Jaccard containment are all reasonable choices here. We found the Jaccard similarity performs the best in our experiments.

Note that for a large $|\mathbf{V}|$ with millions of values, computing similarity for all pairs of values exhaustively is quadratic and not practical. But in reality most pairs of values do not share any overlap, or $\{c|v_1 \in c, c \in \mathbf{C}\} \cap \{c|v_2 \in c, c \in \mathbf{C}\} = \emptyset$. We use techniques similar to [31, 40] from the all-pair-similarity literature, to efficiently compute only non-zero scores on Map-Reduce.

With relatedness scores between pairs of values, we are ready to produce initial candidate cluster trees, to be used for pruning in the next step. We experimented with various well-known clustering approaches, including single-link, average-link, complete-link, correlation clustering, density-based clustering, etc. We found average-link produces initial clusters of the best quality, likely because it is more robust to anomalous co-occurrence scores, and is used in the first step. Clusters generated by techniques like single-link and correlation clustering are of low quality even as candidate trees.

While this step is conceptually simple, the main challenge is efficiency, because the average-link algorithm is of complexity $O(n^3)$, which is known to be difficult to scale to large data sets (in our graph $n > 10$M). We change the vanilla average-link in two ways to make this feasible. First, instead of finding the closest pair of nodes to merge in each clustering step, we perform batch clustering – at each iteration, we merge all nodes whose scores are over a dynamically determined threshold. The threshold for each iteration can be determined based on fixed step-size (e.g., $\{0.95, 0.9, 0.85, ...\}$), or based on a fixed target that certain fraction of nodes should collapse in the iteration (e.g., a score to ensure that 5% nodes will merge). Although the result of the batch clustering will not be the same as the traditional, one-pair-at-a-time clustering, it should be a close approximation. In the very extreme when we set the step-size to very small values so that only one pair is merged in each iteration, this will indeed reduce to the vanilla average-link clustering.

But the batch version of clustering is still non-trivial, because since the large graph would not fit in memory, merging all nodes whose edge scores are above the threshold in each iteration can in the worst case require $O(m)$ Map-Reduce rounds, where m is the number of nodes to merge in that iteration (e.g., for chain graphs). We overcome this challenge by relaxing the semantics, namely we collapse all edges whose scores are over the threshold in one conceptual step, not considering the order of merge which may affect the final result. This merging step is similar to finding connected components (we conceptually drop edges whose scores are lower than the threshold). We thus adapt the *random mate* algorithm [36] developed for connected components on large graphs for this purpose. In the worst case the required number of Map-Reduce rounds is $O(\log m)$ as opposed to $O(m)$.

Algorithm 1 Exhaustive bottom-up clustering

1: $O_c^0 \leftarrow V$
2: $\theta^0 \leftarrow TopK(E^{i+1}, k), i \leftarrow 0$
3: **while** $\theta^i > 0$ **do**
4: $\quad E_{Merge} \leftarrow \{v_1, v_2 | v_1, v_2, s(v_1, v_2) > \theta^i\}$
5: $\quad O_c^{i+1} \leftarrow ConnectedComponents(O_c^i, E_{Merge})$
6: $\quad \theta^{i+1} \leftarrow TopK(E^{i+1}, k)$
7: $\quad i \leftarrow i + 1$
8: **end while**
9: **return** $\bigcup O_c^i$

Algorithm 1 describes our implementation of these high-level ideas. At each iteration i, the algorithm will maintain a graph of super-nodes O_c^i that is the outcome of the merging from the previous step. We select all edges above the threshold θ_i in that iteration, where the threshold θ_i is computed as the top k-th score among all edge in the current graph (line 2 and 6). The parameter k is essentially the batch size and a larger k (e.g., 1% or 5% of nodes) limits the number of batches required and produces results efficiently. We then perform the merging step using *random mate* algorithm [36] (line 5), resulting in a new graph of super-nodes O_c^{i+1}.

We can upper-bound the complexity of this algorithm in terms of the number of Map-Reduce rounds required.

Theorem 1. *For a chosen a batch size of k, the number of distributed rounds for our algorithm will be* $O(\frac{|\mathbf{V}| \log k}{k})$.

Proof. First, the merging step (line 4) uses parallel connected component finding, which requires $O(\log d)$ rounds in the worst case, where d is the graph diameter. Because d is no greater than the number of edges to merge, k, this requires no greater than $O(\log k)$ rounds. For selecting the k-th highest edge score in order to pick the threshold (line 6), we will utilize the quick select algorithm [28], which is also $O(\log k)$ rounds. Since we need $|\mathbf{V}|/k$ such merging step before all nodes are merged, the total number of rounds is no more than $O(\frac{|\mathbf{V}| \log k}{k})$. □

4 CONCEPT SELECTION BY TREE REDUCTION

We are now ready to describe the next step of tree reduction, which is the focus of this work. We will analyze the problem in Section 4.1, and solve it optimally on Map-Reduce in Section 4.2.

4.1 Problem Formulation

We formulate the tree reduction problem as constrained optimization. Specifically, given the deep clustering tree (O_c, E_c) produced from the previous stage, we want to select a set of nodes $O \subseteq O_c$ that are most likely to be good concepts, subject to certain size constraints.

The first question here is how to determine what nodes in O_c are likely to be good concepts. Our observation is the following: given a large spreadsheet corpus, while some table columns in the corpus are random sub-sets/super-sets of natural concepts, a non-trivial fraction of columns should actually correspond to ideal concepts,

with every entity instance in the concepts and nothing more. The hope is that the corpus will guide us in finding them, because clean columns of important concepts should hopefully occur more often than random sub-sets/super-sets of concepts (which should not be repeatedly occurring if they are random and not good concepts).

More specifically, we use the existence of table column c that is almost identical to a candidate cluster node $o \in O_c$ as an evidence that o may be a good concept. Furthermore, the more such columns we find, the more likely o is a desirable concept. For example, the concept of "all ATUs in the world" repeatedly occur as a column in many spreadsheets, which is likely a better concept than its random subsets. The same is true for ATUs in related geographical locations (same countries or same continents), which also occur frequently and may be good concepts. Therefore, we define the "quality" of a node in the clustering tree $o \in O_c$, as all table columns $C \subset \mathbf{C}$ that closely match the content in node o. We can say that these columns C are "covered" or can be "described" by o. Since we need to select a set of nodes $O \subseteq O_c$, we can also define the collective "quality" of these nodes as the union of the table columns covered by O. This is formalized in the following definition.

Definition 2 (corpus coverage). *A node $o \in O_c$ in the clustering tree can cover a table column $c \in \mathbf{C}$, if the Jaccard similarity between o and c is higher than a predefined threshold τ (e.g., 0.95). We use $D(o) = \{c | c \in \mathbf{C}, Jaccard(o, c) \geq \tau\}$ to denote the columns that node o can cover, which we also refer to as the "corpus coverage" of o.*

For a set of nodes $O \subseteq O_c$, the collective coverage of O, denoted by $D(O)$, is defined as the union of the coverage of individual nodes, or $D(O) = \bigcup_{o \in O} D(o)$.

When a concept o and a column c have high similarity, we refer to this as o can cover c, or interchangeably the concept o can *describe* c in the spirit of *minimum description length*.

Intuitively, we want to make sure that the nodes we select can approximate the original corpus as closely as possible, so that they are likely good concepts. So the coverage $D(O)$ is effectively the objective function of our tree reduction problem we study.

A trivial way to maximize $D(O)$ is to select all nodes in O_c, which however is not interesting as the resulting tree is too big for humans to use (i.e., if we run the previous clustering step for 100 iterations, the tree will have 100 levels of hierarchy which is clearly too big). In this deep and big candidate clustering tree, many nodes are are actually not good concepts, and nodes may be highly redundant to each other (they may only differ slightly due to exhaustive clustering).

So we impose a constraint on the *height* the resulting tree, to ensure that humans can more easily understand and curate the reduced tree, and at the same time the most important nodes are preserved, which should hopefully correspond to good concepts. We choose to constrain the height of the tree instead of the number of nodes, because height is independent of the fan-out of the tree, and directly corresponds to the number of hierarchies in the resulting tree. For the ATU concept, with height set to 3, we may obtain concepts corresponding to ATUs in the same countries, ATUs in the same continents, and all ATUs in the world.

From the input candidate cluster tree (O_c, E_c), it is easy to see that by selecting a subset of nodes $O \subseteq O_c$, it always induces a reduced tree, because ancestor-descendant relationships from (O_c, E_c) are guaranteed to be preserved. This allows us to focus on selecting nodes in O_c without worrying about the structure of the resulting graph.

We therefore formulate our concept selection problem follows. Given an input clustering tree (O_c, E_c) generated from previous step, the table corpus $\mathbf{C}$, a coverage function D, and a height constraint h, select a subset of nodes $O \subseteq O_c$, so that the collective coverage $|D(O)|$ is maximized, while the reduced-tree induced by O has height no more than h. We cast this as a general problem as described below.

Problem 1. *Maximum-Coverage Tree Selection (MCTS). Given an input tree (O, E), a set of targets T against which nodes in O can cover, as defined by a monotonic coverage function $D : 2^O \rightarrow 2^T$, and a height constraint h, select a set of tree nodes, $U \subseteq O$, so that the reduced tree induced by U has height of as most h, and the target coverage $|D(U)|$ is maximized.*

We note that the general idea of the formulation is consistent with well-known principles such as *minimum description length* [37] and *Occam's razor* [24]. It can be shown that this problem is APX-hard using reduction from Set Cover.

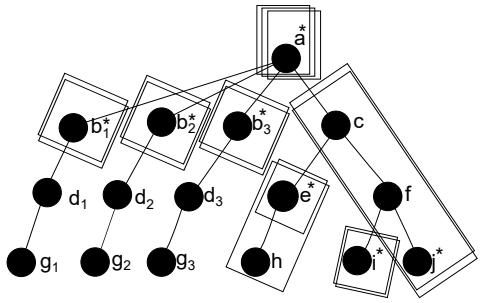


Figure 2: Continuation of the example in Figure 1. Reformatted to illustrate the tree reduction problem of MCTS.

We illustrate the problem using our running example.

Example 2 (MCTS problem). *We continue with Example 1 and redraw the deep clustering tree of Figure 1 in Figure 2. We use a solid dot to represent a node. The root a here again represents all ATUs in the world.*

If a node o in the tree can "cover" a table column in our spreadsheet as defined by the coverage function $D(o)$, we draw a square around that node. The number of squares for each node visually indicates the corpus coverage of that node.

It can be verified that the optimal solution of MCTS on this tree with a height constraint 2 is to select $\{a, b_1, b_2, b_3, e, i, j\}$, which has a total coverage of 15 (each branch may in this case correspond to ATUs in different continents).

4.2 Solve MCTS using Dynamic Programming

In this section we present a dynamic-programming-style algorithm to solve the MCTS problem optimally on Map-Reduce. The challenge is that MCTS is intractable in general, and we need to work with large data sets on Map-Reduce. We efficiently compute and

store partial solutions to subproblems of MCTS that are needed to compute the global optimal solutions.

The subproblem structure can be described as follows.

Problem 2. *Subproblem to MCTS (S-MCTS). Given an instance of the MCTS problem on a tree of nodes V and a height constraint h, denoted as MCTS(V, h), a subproblem is defined with respect to a sub-tree V_S rooted at node $v_S \in V$ and a height constraint $h' \leq h$, written as S-MCTS(V_S, h').*

If MCTS were to have *optimal substructure* [15], it would be natural to solve smaller problems at leaf levels of a tree, and iteratively solve larger problems utilizing solutions at lower levels. However, there is no optimal substructure in MCTS and larger problems cannot be solved optimally that way. Instead, we need to efficiently memorize the set of *promising* partial solutions to subproblems that may be needed to form the global optimal.

Represent targets covered by tree nodes. In order to represent the solution to the subproblems, we first describe how we represent the set of targets (a general term we introduce in Problem 2 that refers to columns in this specific problem) for each tree node. Representing targets naively for each node as a set of target-ids is inefficient and cannot scale to large problems. We efficiently encode the targets of nodes utilizing the tree structure, which is crucial for both time and space efficiency.

We define *tree coverage*, which is the inverse of corpus coverage in Definition 2, by mapping targets (columns) back to tree nodes.

Definition 3 (tree coverage). *For a given MCTS problem, the tree coverage for a target $t \in T$ is defined as $D^{-1}(t) = \{v | v \in V, t \in D(v)\}$.*

We further characterize the targets into three main categories.

Definition 4. *(Path-target, partial path-target, non-path-target). We say a set of nodes $V' \subseteq V$ is a* tree path*, if they form one uninterrupted path in the tree. A target $t \in T$ is a* path target *if its tree coverage $D^{-1}(t)$ is a tree path. A target $t \in T$ is a* partial path target *if its tree coverage $D^{-1}(t)$ is not a tree path but is a subset of a tree path. Finally, a target $t \in T$ is* non-path target *if it is neither a path target nor a partial path target.*

Intuitively, the path target is a target that is covered by a continuous sequence of nodes in one path of the deep clustering tree. In Figure 2, the targets (represented as rectangles) corresponding to the right-most path c-f-j are path-targets; so are others targets (rectangles) as they all form uninterrupted paths in the tree. This is the most common case – as cluster nodes grow larger moving up the tree, nodes that are close enough to a desired target often form an uninterrupted tree path.

In contrast, partial path targets represent targets that are covered by nodes on an interrupted path in a tree. Finally, non-path targets represent targets that are covered by nodes in a tree that is no longer a subset of a tree path. Neither of these two cases are common but can exist in theory. It is worth noting that with a τ value greater than 0.5, the bad case of non-path targets will never occur, which can be shown by contradiction.

We represent these three types of targets differently: for a path target, we can efficiently encode it by remembering the lowest descendant in the its tree path, v_{start}, and the highest ancestor in the tree path, v_{end}. For a partial path target, let P be the smallest tree path that includes this partial path, we can encode the partial path target using v_{start}, v_{end} of path P, plus $V_{missing}$ which are nodes that are absent from P. Finally for non-path targets, we simply map each node to a set of these target-ids.

Physically, given an input tree in an MCTS problem, we use a dictionary $dict_L$, keyed by v_{start}, v_{end}, to store the number of all path targets; and we use a dictionary $dict_{PL}$, keyed by $v_{start}, v_{end}, V_{missing}$, to store the number for all partial path targets. For non path targets, we directly store the set of targets for each node $v \in V$, denoted as $covered_{NL}(v)$. Note that for non-path targets it essentially degenerates to the naive approach of storing node coverage. But because almost all targets are path-targets or partial-path-targets (i.e., with $\tau > 0.5$ non-path targets are guaranteed to be empty as discussed before), we can encode targets very efficiently.

Represent a solution to subprogram S-MCTS. For a given instance of S-MCTS(V_S, h) problem, because the problem lacks *optimal substructure*, naively we will need to memorize all possible solutions, each of which consists of a set of tree nodes $S \in V_S$, and the corresponding set of targets that S can cover, namely $D(S)$.

But for reasons that will be clear soon, conceptually we only need three pieces of information for each solution: (a) the set of nodes selected $S \in V_S$ (whose induced tree has a height $H(S) \leq h$); (b) the total number of targets covered by S, $|D(S)|$, also denoted as *covered*; (c) the set of targets that may be covered in the future that are not already covered by S, denoted as $D(V \setminus V_S) \setminus D(S)$. The last part in (c) can be efficiently encoded using the data structure of $dict_L$, $dict_{PL}$, and $covered_{NL}$ as described above.

Derive the set of Promising Solutions to subprogram S-MCTS. For a given S-MCTS(V_S, h) problem, we now show how to use the three pieces of information in a solution above to prune away unpromising solutions that can never be part of a global optimal in a larger problem, which is critical to the efficient induction of the global optimal.

Let the set of all solutions to S-MCTS(V_S, h) be $PST_{v,h}$, where v is the root of V_S. We can prune away unpromising solutions in $PST_{v,h}$, by defining a partial order $<_S$ over solutions. Suppose we have two solutions, S_1 and S_2, we say a solution S_1 is inferior to another solution S_2, $S_1 <_S S_2$, if S_1 will never be a better choice than S_2 in any possible way. Specifically, in the best case for S_1, on top of all targets already covered by S_1 (the number *covered* defined above), if we include all path and partial-path targets not already covered, whose $v_{start} \in V_S$ but $v_{end} \notin V_S$, and if the resulting sum is still less than the sum of the covered path and partial path targets in S_2 (can be computed from $dict_L$, $dict_{PL}$), then we can show that S_1 is always inferior to S_2 as a part of solutions to larger problems S-MCTS$(\overline{V}_S, h')$, with $\overline{V}_S \supset V_S$ and $h' \geq h$. We can thus discard S_1 without affecting the optimality of our recursive induction.

The overall induction for node v and height constraint h can be expressed as below. Let $\{j_1, j_2, \ldots, j_{k_v}\}$ be v's children nodes, $PST \times PST'$ be the cross-product of two sets of solutions (by union

solutions from each set). The desired $PST_{v,h}$ can be obtained as

$$S_1 = \{includeCurrent(l_1, l_2, \ldots, l_{k_v}) | (l_1, l_2, \ldots, l_{k_v}) \in PST_{j_1,h-1} \times PST_{j_2,h-1} \times PST_{j_{k_v},h-1}\} \quad (1)$$

$$S_2 = \{notIncludeCurrent(l_1, l_2, \ldots, l_{k_v}) | (l_1, l_2, \ldots, l_{k_v}) \in PST_{j_1,h} \times PST_{j_2,h} \times PST_{j_{k_v},h}\} \quad (2)$$

$$PST_{v,h} = prune(S_1 \cup S_2) \quad (3)$$

In case (1), the function *includeCurrent* corresponds to the scenario of including v in solution, in addition to selecting best solutions from each of v's children independently using constraint $h-1$. In case (2), the function *notIncludeCurrent* corresponds to not including v in the solution, but selecting best solutions from each of v's children with constraint h. The function *prune* corresponds to pruning away solutions dominated by others using the partial order $<_S$ defined before.

A more detailed description of the algorithm is shown in Algorithm 2. We first perform initialization to obtain *PST* for leaf nodes, in which case the incoming results from children nodes are empty (line 3-9). During the main body of dynamic programming induction (line 10-27), we proceed from nodes of lower tree height to higher. For each node we obtain *PST* for all height constraints, using the induction procedure above. There are two possible cases as discussed above: to include the current node into the solution or to keep the solution as is. For the first case, the derivation process would be: 1) A $derive\_from\_children$ operation that sums up the corresponding elements in $dict_L$, $dict_{PL}$, $covered$ for all the input solutions from children nodes. It also needs to merge the set $covered_{NL}$ in addition. 2) A $maintain\_active\_targets$ operation to add entries in $dict_L$, $dict_{PL}$, that have the current node as v_{start}, and remove entries in $dict_L$, $dict_{PL}$ that have the current node as v_{end}. 3) A $cover\_targets$ operation to remove all entries in $dict_L$, $dict_{PL}$ except for those in $dict_{PL}$ having current node among $V_{missing}$, and sum them up into *covered*. If we removed entries from step 2, we also need to add them into *covered* here. Furthermore, we will merge the newly covered non path targets into $covered_{NL}$ and update *covered* based on that. For the second case, the derivation process will be just step 1 and step 2 as in the first case.

Finally, we will obtain the *PST* for the root of tree with height constraint h. We can then select the solution of height h that has the highest target coverage as the final result R.

The simple example below shows the induction hierarchy for Figure 2.

Example 3 (MCTS algorithm). *We revisit Example 2 in Figure 2, where the optimal solution with height constraint 2 is $\{a, b_1, b_2, b_3, e, i, j\}$, which has a total score of 15. This is formed by combining one solution each from $PST_{b_3,1}$, $PST_{b_2,1}$ $PST_{b_1,1}$, and $PST_{c,1}$, respectively. For the solution from $PST_{b_3,1}$, it comes from a solution in $PST_{b_2,0}$, which traces back to a solution in $PST_{b_1,0}$. Same is true for $PST_{b_2,1}$ and $PST_{b_1,1}$. For the solution in $PST_{c,1}$, it comes from combining a solution in $PST_{e,1}$, which in turn traces back to $PST_{h,0}$; and $PST_{f,1}$, which traces back to $PST_{i,1}$ and $PST_{j,1}$. It can be verified that these solutions are not dominated by others in the induction process. From these traces we can recover the optimal nodes selection for this problem.*

Algorithm 2 Algorithm for MCTS

1: INPUT: tree (V, E), target set T, target coverage function $d : V \rightarrow 2^T$, height constraint h
2: OUTPUT: selected nodes $V_S \subseteq V$
3: **for** node v with node height 0 **do**
4: $T \leftarrow add\_newly\_active\_targets(v)$
5: **if** the combination comes from height constraint $h-1$ **then**
6: $T \leftarrow cover\_active\_target(T, v)$
7: **end if**
8: $ST \leftarrow discard\_inactive\_targets(T, v)$
9: **end for**
10: **for** *stage* from 0 up to the height of the original tree **do**
11: **for** node v with node height *stage* **do**
12: $S = \emptyset$
13: $\{j_1, j_2, \ldots, j_{k_v}\} \leftarrow$ children of v
14: **for** height constraint $h = 0 \ldots H$ **do**
15: **for** $(ST_1, ST_2, \ldots, ST_{k_v}) \in PST_{j_1,h-1} \times PST_{j_2,h-1} \times PST_{j_{k_v},h-1} \cup PST_{j_1,h} \times PST_{j_2,h} \times PST_{j_{k_v},h}$ **do**
16: $ST = derive\_from\_children(ST_1, ST_2, \ldots, ST_{k_v})$
17: $ST = maintain\_active\_targets(ST, v)$
18: **if** the combination comes from height constraint $h-1$ **then**
19: $ST = cover\_targets(ST, v)$
20: **end if**
21: $S \leftarrow S \cup ST$
22: **end for**
23: $PST_{v,h} \leftarrow \{ST | ST \in S, \nexists ST' \neq ST, ST <_S ST'\}$
24: **end for**
25: **end for**
26: **end for**
27: $ST_{OPT} \leftarrow \{ST | ST \in PST_{\text{root of tree}(V,E),H}, |ST$ has the highest *covered* value $\}$
28: $R \leftarrow backtrack(ST, V, E, \{PST(v, h)\})$
29: **return** R

The algorithm is parallelizable since the computation in each for-loop is independent of others.

5 EXPERIMENT

In this section we present experimental evaluations of the proposed method. Our goal is to (1) evaluate the quality of concepts produced by different approaches; (2) understand the effectiveness of tree reduction approach; and (3) measure sensitivity of results to different parameters.

5.1 Experimental Setup

5.1.1 Spreadsheet Corpus.
We perform the concept discovery on a corpus of spreadsheet tables crawled from Microsoft intranet. There are a total of over 500K tables, 3.2M table columns, 13M distinct cell values, and over 2B edges for pairs of values with non-zero co-occurrence. Many entities and concepts are enterprise-specific, examples of which are shown in Table 1.

5.1.2 Computing Environment.
We conduct our experiments on Microsoft's production Map-Reduce clusters [12] alongside with other production jobs. The first stage of

candidate cluster generation is more expensive but takes no more than 24 hours in our experiments, and the second stage of concept selection takes no more than 4 hours. We find these to be acceptable since these jobs can run overnight in an offline setting, where latency is not critical. We would like to note that the resulting hierarchical concept trees are small enough to be explored/manipulated by human users at interactive speed.

5.1.3 Quality Evaluation.

We constructed 100 ground truth concepts with the following procedure: We randomly sample from the spreadsheets a set of columns that is manually judged to be clean and relatively close to complete. We then search in the spreadsheet corpus to identify additional table columns that share significant overlap with these initial columns, and if these additional columns are judged to be part of the ground truth we will merge them and prune away outliers, until we are reasonably confident that the concepts are complete.

We measure quality of produced clusters against these ground truth concepts using F-scores. For each ground truth concept, we will find a cluster generated by an algorithm with the highest F1-score, and we will select that cluster as the prediction by the algorithm. Finally we report average F1-scores across all ground truth concepts.

5.1.4 Methods Compared.

Correlation Clustering. We implement the Map-Reduced-based correlation clustering algorithm [14] and evaluate the quality of the resulting clusters. This algorithm works by sampling pivot nodes and growing clusters around them in iterations.

Density-based Clustering. Density-based clustering such as DB-SCAN [20] is a popular method for clustering. We implement the DB-SCAN on Map-Reduce: in each round, we perform a *reduce* to compute the density and neighborhood size of each node to determine whether that node is a core or edge. We then iteratively find the reachable region of each core to merge. The minimum size of clusters is set to 5.

Connected Components. We additionally experiment with a simple connectivity based method on the co-occurrence graph to determine concept boundaries. In this method, we simply merge all nodes whose edge scores is above a certain threshold, which is also equivalent to Single-Link clustering. We use the algorithm discussed in [36] on Map-Reduce.

Complete Linkage. We also test the complete linkage clustering, where a pair of nodes merge only when all constituent nodes are similar enough to each other.

WebSets. We implement and compare with the WebSets approach proposed in [18], which is closely related to the problem we study. The WebSets method emits for each table a set of consecutive triplets of entities (e.g., India, China, Canada), and then merge the triplets together if they share significant overlap in entities. We use the same parameters as in [18].

C^4. The method proposed in this work is denoted as C^4, which is short for Concept Construction from Coherent Clusters. In the first stage of cluster generation, we run 20 iterations of hierarchical clustering, resulting in a candidate tree with height 20. In the second stage of tree reduction, we set the height constraint to 1 and 3. The corresponding results are denoted as C^4-H1, and C^4-H3, respectively. We also test quality on the initial cluster tree generated from the first step, which is denoted as C^4-All. Note that

Figure 3: Overall quality comparison (best parameters)

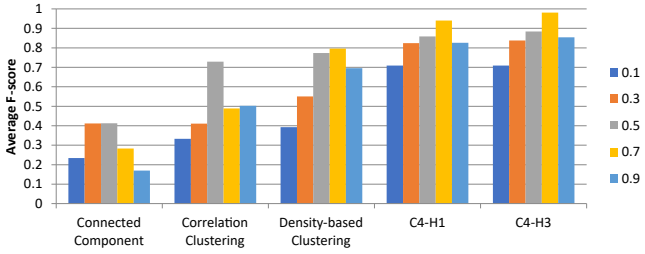


Figure 4: Sensitivity of quality to different parameters

when using a height constraint of 1, the result in C^4-H1 effectively is a disjoint set of nodes (as opposed to a hierarchical tree), which is directly comparable to standard clustering method. For trees with height greater than 1 (e.g., C^4-H3), we produce hierarchical trees with overlapping concepts, which essentially allows them to make more than 1 prediction for the same concept, and in turn makes the results more favorable compared to standard clustering.

5.2 Quality Evaluation

The overall quality comparison can be found in Figure 3. We tuned parameters for all methods reported here (e.g., thresholds for clustering), and only show the best results for each method in this figure.

We observe that C^4-H1 outperforms all existing methods we implemented by at least 10 percentage points. C^4-H3 and C^4-All have even higher F-score, but as we discussed above, these results are not directly comparable to standard clustering, because for height constraints greater than 1 we produce hierarchical trees that contain overlapping concepts. It is interesting to note, however, that from C^4-All (20 levels) to C^4-H3 (3 levels), our tree reduction produces a smaller tree with virtually no difference in quality. This is encouraging as it shows the effectiveness of the proposed tree reduction approach. WebSets does not perform well in this test, partly because it requires triples to be in specific order before merging, which is often too strict to form complete concepts.

In Figure 4 we analyze the sensitivity of each method to threshold parameters. The exact interpretations of thresholds in each method are slightly different – in the case of clustering they indicate stopping criteria, whereas in C^4 this is the parameter τ used in Definition 2. For C^4-H1 and C^4-H3, using Jaccard similarity of 0.7 achieves the best performance.

We report the quality of individual concepts in Figure 5. C^4 achieves high accuracy for most concepts. Results with different

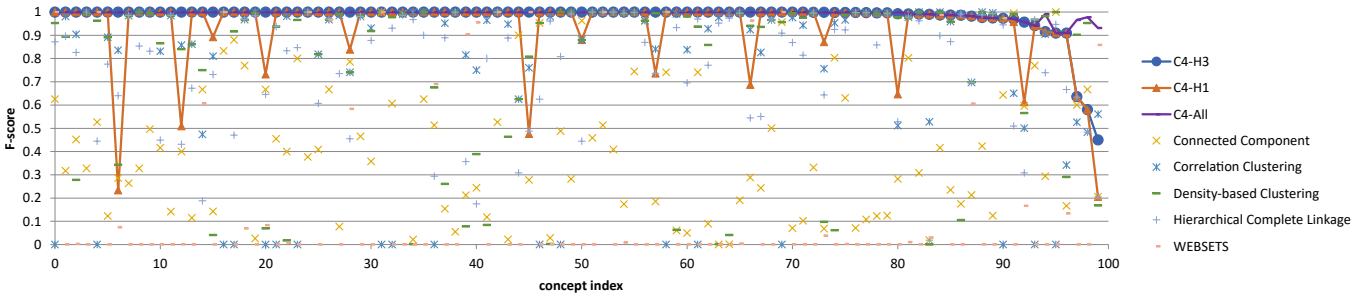


Figure 5: Performance by individual concepts

height constraints suggest for almost all concepts, reducing the initial candidate cluster tree to 3 levels (C^4-H3) incurs virtually no cost in terms of quality. Further reducing the tree height to 1 (C^4-H1), however, does lead noticeable drop in quality for 13 out of 100 concepts.

Validate corpus coverage as a quality measure. A key aspect of our problem formulation in Section 4.1 is to use corpus coverage as a proxy of quality in selecting concept nodes. The idea is that if a small set of concepts can cover/describe a large set of columns, then they are likely to be useful concepts. Intuitively, this gives us a more flexible way to select concepts – nodes are selected directly based on table data, as opposed to standard clustering that always terminates at a fixed threshold.

In Table 2 we analyze the number of concepts generated by different methods, and the number of spreadsheet table columns that can be covered/described by these concepts. A method is intuitively more desirable if it uses a few concepts to cover a larger number of table columns. From Table 2 we can see that the ratio between these two numbers indeed correlate well with the quality results obtained using manually-labeled benchmarks. We can see that C^4-H3 and C^4-H1 score well in this regard, which is not surprising since this is the optimization objective in the problem formulation. It is interesting to note that C^4-H3 uses less than 1% of the clusters of C^4-All, without losing much coverage of columns.

We think this can be of independent interest to the general problem of data clustering. In standard clustering, predetermined thresholds are used to terminate merging in all tree branches. In practice this may be too rigid as scores computed from different branches are not directly comparable. An alternative is to directly optimize for a more relevant metric specific to the clustering problem (like we did for concept selection), which may provide better clusters compared to using fixed thresholds.

	concept generated	table columns covered
Connected Component	189K	4K
Correlation Clustering	39K	23K
Density Clustering	59K	26K
Hierarchical Complete Linkage	461K	19K
WebSets	737	2134
C^4-H1	16K	36K
C^4-H3	19K	43K
C^4-All	3641K	53K

Table 2: Analysis of corpus coverage for different methods.

6 RELATED WORK

Knowledge bases such as YAGO [42], Freebase [7], and DBPedia [3] are widely used today in a variety of applications, and embody really impressive advances in this field. With a few exceptions (e.g. [43]), most techniques focus on the public web domain as opposed to proprietary enterprise domains, and are based on variations of text-patterns to construct concepts [2, 30, 34, 43, 49].

The topic of constructing concepts on enterprise data is large unexplored to this date, partly because of the difficulty of the task attributable to the scarcity of text documents in enterprises. Our approach takes the first steps towards this direction, by leveraging spreadsheet tables that are abundant in enterprise and can be close to ideal concepts.

There is an interesting line of work on identifying new entities from tables to enrich existing knowledge bases in the public domain (e.g., YAGO or DBPedia) [6, 19, 35, 39, 50]. Unfortunately, in enterprise domains there is no such preexisting knowledge bases that one can leverage, making these techniques inapplicable in an enterprise setting.

A related problem studied in [48] is to discover mappings relationships from table corpus. Mapping tables are two-column tables satisfying functional dependencies that can be viewed as a specific type of knowledge, which is however largely orthogonal to the concept hierachies studied in this work.

Set-expansion and concept-expansion [10, 22, 27, 47] is another line of related work. In these methods, a small number of seed entity instances and/or a concept name are given as input to algorithms; web tables are then leveraged to automatically discover additional entities in the same concept. These techniques are intended for human users to complete known concepts, one concept at a time. In comparison, the approach studied in this work attempts to discover all concepts simultaneously based on a spreadsheet corpus in an offline manner. While the results of our approach are still intended for humans to curate just like the set-expansion approaches, the offline setting should significantly reduce human overhead of providing concept names and seed entities.

In the context of semantic web, techniques have been developed to map relational database into RDF data and OWL ontologies [11, 41, 45], using a small set of authoritative databases. While these techniques are important, in enterprises it is often difficult to identify a comprehensive set of master databases, since they are likely in different silos and are not easily discoverable/accessible.

In fact, if a small set of master databases were known, then our problem of extracting concepts/entities is trivial. Our approach is built on the observation that comprehensive master databases are not available, and instead solves the concept discovery problem using spreadsheets, which are readily available and in enterprise intranets in large quantities.

In addition to enriching knowledge bases, relational tables are rich data assets that have empowered a variety of application, including data integration [9, 26], synonyms discovery [8, 25], and table search [8, 13], among other things. Leveraging structured tables for data-driven application is a rich area for future research.

7 CONCLUSION AND FUTURE WORK

In this work, we take first steps towards discovering concepts using enterprise spreadsheet tables. We show that it is a promising direction by leveraging a large table corpus. There are a few interesting areas that warrant future research. First, it would be useful to test the proposed method using different table corpora, e.g., on spreadsheets from different enterprises, and on tables from the public web domain. Understanding how to best help users to curate the automatically generated concept hierarchy is another important area of future work.

Acknowledgement. Keqian Li's work was partially supported by NSF IIS 0954125. We thank four annonymous reviewers whose detailed comments were instrumental for revising this paper.

REFERENCES

[1] Google sets. http://labs.google.com/sets.
[2] E. Agichtein and L. Gravano. Snowball: Extracting relations from large plain-text collections. In *DL*, pages 85–94. ACM, 2000.
[3] S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak, and Z. Ives. Dbpedia: A nucleus for a web of open data. In *ISWC*, 2007.
[4] M. Banko, M. J. Cafarella, S. Soderland, M. Broadhead, and O. Etzioni. Open information extraction from the web. In *IJCAI*, 2007.
[5] R. J. Bayardo, Y. Ma, and R. Srikant. Scaling up all pairs similarity search. In *WWW*, 2007.
[6] C. S. Bhagavatula, T. Noraset, and D. Downey. Tabel: Entity linking in web tables. In *ISWC*, 2015.
[7] K. Bollacker, C. Evans, P. Paritosh, T. Sturge, and J. Taylor. Freebase: a collaboratively created graph database for structuring human knowledge. In *SIGMOD*, pages 1247–1250. ACM, 2008.
[8] M. J. Cafarella, A. Halevy, D. Z. Wang, E. Wu, and Y. Zhang. Webtables: exploring the power of tables on the web. *Proceedings of the VLDB Endowment*, (1), 2008.
[9] M. J. Cafarella, A. Y. Halevy, and N. Khoussainova. Data integration for the relational web. In *PVLDB*, 2009.
[10] A. Carlson, J. Betteridge, B. Kisiel, B. Settles, E. R. Hruschka Jr, and T. M. Mitchell. Toward an architecture for never-ending language learning. In *AAAI*, volume 5, page 3, 2010.
[11] F. Cerbah. Learning highly structured semantic repositories from relational databases: the rdbtoonto tool. In *ESWC*, 2008.
[12] R. Chaiken, B. Jenkins, P.-A. Larson, B. Ramsey, D. Shakib, S. Weaver, and J. Zhou. Scope: Easy and efficient parallel processing of massive data sets. In *VLDB*, 2008.
[13] K. Chakrabarti, S. Chaudhuri, Z. Chen, K. Ganjam, and Y. He. Data Services Leveraging Bing's Data Assets. In *IEEE Data Engineering Bulletin*, 2016.
[14] F. Chierichetti, N. Dalvi, and R. Kumar. Correlation clustering in mapreduce. In *KDD*, pages 641–650. ACM, 2014.
[15] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms (3rd ed.)*. MIT Press, 2009.
[16] E. Cortez, P. A. Bernstein, Y. He, and L. Novik. Annotating database schemas to help enterprise search. In *PVLDB*, 2015.
[17] I. F. Cruz and H. Xiao. The role of ontologies in data integration. *JOURNAL OF ENGINEERING INTELLIGENT SYSTEMS*, 2005.
[18] B. B. Dalvi, W. W. Cohen, and J. Callan. Websets: Extracting sets of entities from the web using unsupervised information extraction. In *WSDM*, pages 243–252. ACM, 2012.
[19] X. Dong, E. Gabrilovich, G. Heitz, W. Horn, N. Lao, K. Murphy, T. Strohmann, S. Sun, and W. Zhang. Knowledge vault: A web-scale approach to probabilistic knowledge fusion. In *KDD*, 2014.
[20] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *KDD*, 1996.
[21] O. Etzioni, M. Cafarella, D. Downey, S. Kok, A.-M. Popescu, T. Shaked, S. Soderland, D. S. Weld, and A. Yates. Web-scale information extraction in knowitall. In *WWW*, 2004.
[22] O. Etzioni, M. Cafarella, D. Downey, A.-M. Popescu, T. Shaked, S. Soderland, D. S. Weld, and A. Yates. Unsupervised named-entity extraction from the web: An experimental study. *Artificial intelligence*, (1), 2005.
[23] S. G. and G. J.A. Ontological profiles in enterprise search. *Knowledge Engineering: Practice and Patterns.*, 2008.
[24] H. G. Gauch. *Scientific Method in Practice*. Cambridge University Press, 2003.
[25] Y. He, K. Chakrabarti, T. Cheng, and T. Tylenda. Automatic Discovery of Attribute Synonyms Using Query Logs and Table Corpora. In *WWW*, 2016.
[26] Y. He, K. Ganjam, and X. Chu. SEMA-JOIN: Joining Semantically-Related Tables Using Big Table Corpora. In *VLDB*, 2015.
[27] Y. He and D. Xin. Seisa: set expansion by iterative similarity aggregation. In *WWW*, pages 427–436. ACM, 2011.
[28] C. A. Hoare. Algorithm 65: find. *Communications of the ACM*, 4(7), 1961.
[29] J. Krishnamurthy and T. M. Mitchell. Which noun phrases denote which concepts? In *ACL*, 2011.
[30] D. Lin and P. Pantel. Concept discovery from text. In *international conference on Computational linguistics*, pages 1–7. Association for Computational Linguistics, 2002.
[31] A. Metwally and C. Faloutsos. V-smart-join: A scalable mapreduce framework for all-pair similarity joins of multisets and vectors. In *VLDB*, 2012.
[32] G. A. Miller. Wordnet: a lexical database for english. *Communications of the ACM*, 38(11):39–41, 1995.
[33] A. Passadore, A. Grosso, and A. Boccalatte. Agentseeker: an ontology-based enterprise search engine. In *MALLOW*, 2009.
[34] S. P. Ponzetto and M. Strube. Deriving a large scale taxonomy from wikipedia. In *AAAI*, volume 7, pages 1440–1445, 2007.
[35] G. Quercini and C. Reynaud. Entity discovery and annotation in tables. In *Proceedings of the 16th International Conference on Extending Database Technology*, pages 693–704. ACM, 2013.
[36] V. Rastogi, A. Machanavajjhala, L. Chitnis, and A. Das Sarma. Finding connected components in map-reduce in logarithmic rounds. In *ICDE*, pages 50–61. IEEE, 2013.
[37] J. Rissanen. Modeling by shortest data description. *Automatica*, 1978.
[38] D. Ritze, O. Lehmberg, Y. Oulabi, and C. Bizer. Profiling the potential of web tables for augmenting cross-domain knowledge bases. In *WWW*, pages 251–261. International World Wide Web Conferences Steering Committee, 2016.
[39] D. Ritze, O. Lehmberg, Y. Oulabi, and C. Bizer. Profiling the potential of web tables for augmenting cross-domain knowledge bases. In *WWW*, 2016.
[40] A. D. Sarma, Y. He, and S. Chaudhur. Clusterjoin: A similarity joins framework using map-reduce. In *VLDB*, 2014.
[41] J. F. Sequeda, M. Arenas, and D. P. Miranker. On directly mapping relational databases to rdf and owl. In *WWW*, 2012.
[42] F. M. Suchanek, G. Kasneci, and G. Weikum. Yago: a core of semantic knowledge. In *WWW*, pages 697–706. ACM, 2007.
[43] F. Tao, B. Zhao, A. Fuxman, Y. Li, and J. Han. Leveraging pattern semantics for extracting entities in enterprises. In *WWW*, pages 1078–1088. International World Wide Web Conferences Steering Committee, 2015.
[44] T. Tran, P. Cimiano, S. Rudolph, and R. Studer. Ontology-based interpretation of keywords for semantic search. In *ISWC*, 2007.
[45] R. D. Virgilio, A. Maccioni, and R. Torlone. R2g: a tool for migrating relations to graphs. In *EDBT*, 2014.
[46] C. Wang, K. Chakrabarti, Y. He, K. Ganjam, Z. Chen, and P. A. Bernstein. Concept expansion using web tables. In *WWW*, pages 1198–1208. International World Wide Web Conferences Steering Committee, 2015.
[47] R. C. Wang and W. W. Cohen. Iterative set expansion of named entities using the web. In *ICDM*, pages 1091–1096. IEEE, 2008.
[48] Y. Wang and Y. He. Synthesizing mapping relationships using table corpus. In *SIGMOD*, 2017.
[49] W. Wu, H. Li, H. Wang, and K. Q. Zhu. Probase: A probabilistic taxonomy for text understanding. In *SIGMOD*, pages 481–492. ACM, 2012.
[50] X. Zhang, Y. Chen, J. Chen, X. Du, and L. Zou. Mapping entity-attribute web tables to web-scale knowledge bases. In *DASFAA*, 2013.