Cost-Effective Conceptual Design Over Taxonomies

Ali Vakilian
Massachusetts Institute of Technology
Cambridge, MA
vakilian@mit.edu

Yodsawalai Chodpathumwan*
University of Illinois at Urbana-Champaign
Urbana, IL
ychodpa2@illinois.edu

Arash Termehchy
Oregon State University
Corvallis, OR
termehca@oregonstate.edu

Amir Nayyeri
Oregon State University
Corvallis, OR
nayyeria@oregonstate.edu

ABSTRACT

It is known that annotating entities in unstructured and semi-structured datasets by their concepts improves the effectiveness of answering queries over these datasets. Ideally, one would like to annotate entities of all relevant concepts in a dataset. However, it takes substantial time and computational resources to annotate concepts in large datasets and an organization may have sufficient resources to annotate only a subset of relevant concepts. Clearly, it would like to annotate a subset of concepts that provides the most effective answers to queries over the dataset. We propose a formal framework that quantifies the amount by which annotating entities of concepts from a taxonomy in a dataset improves the effectiveness of answering queries over the dataset. Because the problem is NP-hard, we propose an efficient approximation for the problem. Our extensive empirical studies validate our framework and show the accuracy and efficiency of our algorithm.

ACM Reference format:
Ali Vakilian, Yodsawalai Chodpathumwan, Arash Termehchy, and Amir Nayyeri. 2017. Cost-Effective Conceptual Design Over Taxonomies. In *Proceedings of 20th International Workshop on Web and Databases, Chicago, Illinois, USA, May 2017 (WebDB'17),* 7 pages.
DOI:

1 INTRODUCTION

Taxonomies provide shared understandings of concepts in domains of interest [1]. In particular, they facilitate query answering over unstructured and semi-structured datasets in these domains. For example, assume that a user likes to find information about types of pains caused by Trachoma over

```
<article>
 Granular conjunctivitis causes pain in the outer surface or cornea.
</article>
<article>
 Stye may lead to pain on the eyelids.
</article>
<article>
 GAS caused infections cause pain in tissues.
</article>
```

Figure 1: Medical article excerpts.

excerpts of the medical articles in Figure 1. In the absence of any structured data, she may explore this dataset using inherently ambiguous keyword queries and submit query Q_1:*Trachoma pain*. Unfortunately, the article about Trachoma in Figure 1 refers to this infection by its other name, *Granular conjunctivitis*. Because all articles contain the term *pain*, the query interface may return all articles, two of which do not have any information about Trachoma.

Given a taxonomy, we can annotate entities in an unstructured dataset by their concepts in the taxonomy. Users may also learn the taxonomy and use its concepts in their queries. Figure 3 depicts fragments of the Medical Subject Heading (MeSH) taxonomy, in which nodes denote concepts and edges show subclass relationships [6]. Figure 2 shows the medical article excerpts in Figure 1 whose entities are annotated by their concepts from MeSH taxonomy. Now, our user may mention the concept *Trachoma* in her query and the query interface will return only the articles that contain entities from this concept.

Organizations often use available taxonomies to annotate their datasets so that more users can effectively search and explore their data. For example, the U.S. National Library of Medicine annotates the articles in MEDLINE/PubMED using concepts in the MeSH taxonomy [6]. Many organizations also use taxonomies to extract entities in general domains. For instance, researchers have used the ProBase taxonomy to extract concepts in general domains from Web data [15]. Also, Google and Bing ask organizations to annotate their Web documents by concepts in Schema.org taxonomy, which is developed for datasets in general domains.

Ideally, one would like to annotate the instances of all concepts in a given taxonomy from a dataset to answer all queries effectively. An organization has to spend time, financial and computational resources, and manual labor to accurately annotate entities of a concept in a large dataset [3, 4, 8, 13]. It usually has to develop or obtain a complex program called a *concept annotator* to annotate instances of

*The first two authors have equally contributed to the paper.

WebDB'17, Chicago, Illinois, USA

DOI:

```
<article>
 <Trachoma>Granular conjunctivitis</Trachoma> causes pain in the
 outer surface or cornea.
</article>
<article>
 <Hordeolum>Stye</Hordeolum> may lead to pain on the eyelids.
</article>
<article>
 <Ecthyma>GAS caused infections</Ecthyma> cause pain in tissues.
</article>
```

Figure 2: Annotated medical article excerpts.

a concept in a dataset. One may also use machine learning algorithms to develop an extractor for a concept to avoid hand-tuned programming rules. However, these machine learning algorithms need relevant features which are usually unclear from the dataset. Developers have to find the relevant features through trial and error over numerous iterations [2]. Furthermore, the structure and content of underlying datasets evolve over time; annotators should be regularly rewritten and repaired.

Because of limited financial or computational resources, an organization may not be able to afford to develop and maintain annotator for all concepts in a taxonomy. Also, many users may need an annotated dataset quickly and cannot wait days to get an updated and fully annotated collections. It may afford to annotate only a subset of concepts in a taxonomy. Similarly, many users may not have the time to learn all concepts in a large taxonomy and may prefer to learn and use a relatively small subset of the taxonomy in their queries. For example, an organization may annotate entities in Figure 1 with only concepts *Eye-Infection* and *Skin-Infection* from MeSH taxonomy and get the collection in Figure 4. Intuitively, a query interface may provide less effective answers to queries over the dataset in Figure 4 than the one in Figure 2. Assume that a user wants to find information about the type of pain associated with Trachoma. She may mention the concept *Eye-Infection* in her query. The query interface may return the articles about Trachoma and the one about Hordeolum. Nevertheless, the annotation in Figure 4 still helps the query interface not to return the non-relevant article about the skin infection. Clearly, we would like to select a subset of concepts whose required time and/or resources for annotation do not exceed our budget and most improves the effectiveness of answering queries.

Currently, concept annotation experts have to use their intuitions to find cost-effective subsets of concepts from an input taxonomy. We call this problem the Cost-Effective Conceptual Design (CECD) and make the following contributions. We develop a framework that quantifies the amount of improvement in the effectiveness of answering queries by annotating the dataset by a subset of concepts from a taxonomy in Section 3. We also formally define the CECD problem over tree-shaped taxonomies in Section 3. As the CECD problem is NP-hard, we propose an efficient approximation algorithm for it in Section 4. We evaluate the accuracy of our framework, and effectiveness and efficiency of our algorithm using a large real-world dataset, real-world taxonomies, and a sample of a real-world query workload in Section 5. The proofs of our theorems can be found in [14].

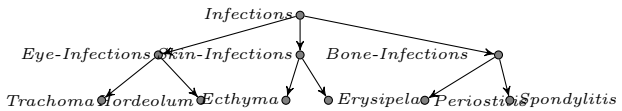

Figure 3: Fragments of MeSH taxonomy

```
<article>
 <Eye-Infections>Granular conjunctivitis</Eye-
Infections> causes pain
 in the the outer surface or cornea.
</article>
<article>
 <Eye-Infections>Stye</Eye-Infections> may lead to pain on the eyelids.
</article>
<article>
 <Skin-Infections>GAS caused infections</Skin-
Infections> cause pain
 in tissues.
</article>
```

Figure 4: Medical article excerpts annotated with more general concepts.

2 RELATED WORK

In our prior work, we have examined the problem of selecting cost effective designs from an unorganized set of concepts for annotation [13]. Nevertheless, the concepts in most real-world domains are maintained in taxonomies rather than unorganized sets. As the framework proposed in [13] does not consider superclass/subclass relationships between concepts, it cannot measure the effectiveness gained by designs over taxonomies. Because taxonomies have richer structures than unorganized sets of concepts, they provide new opportunities and challenges for finding cost-effective conceptual designs. We show in Section 3.4 that because the algorithms in [13] do not consider the structure of taxonomy, they select the designs that provide very ineffective answers. Our empirical results over real-world datasets in Section 5.2 also indicate that the algorithms in [13] deliver considerably less effective designs than the ones that take the structure of the taxonomy into account.

There is a large body of work on building large-scale data management systems for annotating and extracting entities and relationships from unstructured data sources [2, 3, 5, 15]. In particular, researchers have proposed several techniques to optimize the running time and/or required computational power of concept annotation programs by processing only a subset of the underlying collection that is more likely to contain mentions to entities of a given concept [4]. Our work complements these efforts by finding a cost-effective set of concepts in the design phase rather than a set of relevant documents in query time.

3 COST-EFFECTIVE DESIGN

3.1 Basic Definitions

Similar to previous works, we do not rigorously define the notion of named entity [1]. We define a named entity (entity for short) as a unique name in some (possibly infinite) domain. A concept is a set of entities, i.e., its instances. Some examples of concepts are *person* and *country*. An entity of concept *person* is *Albert Einstein*. Concept C is a *subclass* of concept D iff we have $C \subset D$. We call D a *superclass* of C. For example, *person* is a superclass of *scientist*. We define *taxonomy* $\mathcal{X} = (R, \mathcal{C}, \mathcal{R})$ as a rooted tree, with *root*

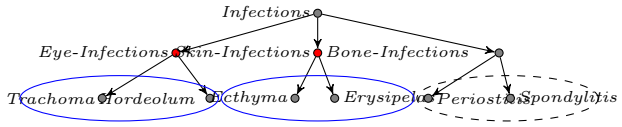

Figure 5: The concepts in red, *Eye-Infections* and *Skin-Infections*, denote the design. The blue curves denote the partitions of design and the dashed curve shows the free concepts of the design.

concept R, vertex set $\mathcal{C}$ and edge set $\mathcal{R}$. $\mathcal{C}$ is a finite set of concepts. For $C, D \in \mathcal{C}$ we have $(C, D) \in \mathcal{R}$ iff D is a subclass of C. Every concept in $\mathcal{C}$ that is not a superclass of any other concept in $\mathcal{C}$ is a *leaf concept*. The leaf concepts are leaf nodes in taxonomy $\mathcal{X}$. For instance, the concept *Trachoma* is a leaf concept in Figure 3. Let $\texttt{child}(C)$ denote the children of concept C. For simplicity, we assume that $\cup_{D \in \texttt{child}(C)} D = C$ for all concepts C in a taxonomy.

Each dataset is a set of documents. Dataset DS is in the domain of taxonomy $\mathcal{X}$ iff some entities of concepts in $\mathcal{X}$ appear in some documents in DS. For instance, the set of documents in Figure 1 are in the domain of the taxonomy in Figure 3. An entity in $\mathcal{X}$ may appear in several documents and multiple times in a document in a dataset. For brevity, we refer to the occurrences of entities of a concept in a dataset as the occurrences of the concept in the dataset. A query q over DS is a pair (C, T), where $C \in \mathcal{C}$ and T is a set of terms, e.g., *(person, {Michael Jordan})*. Empirical studies on real world query logs indicate that the majority of entity-centric queries refer to a single entity [10].

3.2 Conceptual Design

Conceptual design (*design*) $\mathcal{S}$ over taxonomy $\mathcal{X} = (R, \mathcal{C}, \mathcal{R})$ is a non-empty subset of $\mathcal{C} \setminus \{R\}$. A design divides the set of leaf nodes in $\mathcal{C}$ into some subsets defined as follows.

Definition 3.1. Let $\mathcal{S}$ be a design over taxonomy $\mathcal{X} = (R, \mathcal{C}, \mathcal{R})$ and $C \in \mathcal{S}$. The partition of C, denoted as $\texttt{part}(C)$, is a subset of leaf nodes in $\mathcal{C}$ such that for all $D \in \texttt{part}(C)$, the lowest ancestor of D **in** $\mathcal{S}$ is C or $D = C$.

Consider the taxonomy described in Figure 5. Let design $\mathcal{S}$ be {*Eye-Infections, Skin-Infections*}. The partitions of $\mathcal{S}$ are $\{Trachoma, Hordeolum\}$ and $\{Ecthyma, Erysipelas\}$. Also, $\texttt{part}$(Eye-Infections) $= \{Trachoma, Hordeolum\}$ and $\texttt{part}$(Skin-Infections) $= \{Ecthyma, Erysipelas\}$. The equality $D = C$ in Definition 3.1 happens where C itself is a leaf node. For each design $\mathcal{S}$, the set of *leaf concepts* that do not belong to any partition are called *free concepts* and denoted as $\texttt{free}(\mathcal{S})$. These concepts neither belong to $\mathcal{S}$ nor are descendant of a concept in $\mathcal{S}$. For example, consider design {*Eye-Infections, Skin-Infections*} over the taxonomy described in Figure 5. The free concepts of $\mathcal{S}$ are {*Periostitis*, *Spondylitis*} as they are not in any partition of $\mathcal{S}$. Let DS be a dataset in the domain of taxonomy $\mathcal{X} = (R, \mathcal{C}, \mathcal{R})$ and $\mathcal{S}$ be a design over $\mathcal{X}$. $\mathcal{S}$ is the design of dataset DS iff for all concepts $C \in \mathcal{S}$, all occurrences of concepts in the partition of C are annotated by C. In this case, we say DS is an *instance* of $\mathcal{S}$. For example, consider the design $\mathcal{T} =$ {*Eye-Infection, Skin-Infection*} over the taxonomy in Figure 3. The dataset in Figure 4 is an instance of $\mathcal{T}$ as all instances of concepts *Trachoma* and *Hordeolum* that belong to the partition of *Eye-Infection*, are annotated by *Eye-Infection* and all instances of concepts *Ecthyma* and *Erysipelas* that are in the partition of *Skin-Infection*, are annotated by *Skin-Infection* in the dataset.

3.3 Design Queriability

Let $\mathcal{Q}$ be a set of queries over dataset DS. Given design $\mathcal{S}$ over taxonomy $\mathcal{X} =(R, \mathcal{C}, \mathcal{R})$, we would like to measure the degree by which $\mathcal{S}$ improves the effectiveness of answering queries in $\mathcal{Q}$ over DS. The value of this function should be larger for the designs that help the query interface to answer a larger number of queries in $\mathcal{Q}$ more effectively. Let the query interface return k candidate answers for query Q in $\mathcal{Q}$ over the unannotated dataset. The effectiveness of returned answers is usually measured using standard metrics of *precision* and *recall* [7]. The precision of the returned list of answers is the fraction of relevant answers for Q in the list. The recall of a returned list of answers is the ratio of the returned relevant answers to the number of total relevant answers for Q in the dataset. Precision-oriented metrics, such as precision at k ($p@k$ for short), are also used to measure the ranking qualities of the results for queries [7]. If a design helps the query interface to replace some non-relevant answers with relevant ones in the returned list for query Q, it improves both the precision and recall of Q in the top k returned answers. Hence, we estimate the amount by which a design increases the fraction of relevant answers in the top k returned answers for Q.

Let $Q : (C, T)$ be a query in $\mathcal{Q}$ such that C belongs to the partition of $P \in \mathcal{S}$. The query interface may consider only the documents that contain information about entities annotated by P to answer Q. For instance, consider query $Q_1 =$ ($Trachoma$, *pain*) over the dataset in Figure 4 whose design is {*Eye-Infections, Skin-Infections*}. The query interface may examine only the entities annotated by *Eye-Infections* in this dataset to answer Q_1. Thus, the query interface will avoid non-relevant results that otherwise may have been placed in the top k answers for Q. The query interface may further rank these answers according to a ranking function, such as the traditional TF-IDF scoring methods [7]. Our model is orthogonal to the ranking scheme of the candidate answers.

Nevertheless, only a fraction of documents with entities annotated by concept P contain information about entities in C. For instance, to answer query $(Trachoma, pain)$ over the dataset in Figure 4, the query interface has to examine all documents that contain instances of concept *Eye-Infections*. Some documents in this set do not have any entity of concept *Trachoma*. We like to estimate the fraction of the results for $Q : (C, T)$ that contains entities of concept C. Given all other conditions are the same, the larger this fraction is, the more likely it is that the query interface delivers more relevant answers in the top k results for Q. Let $d(C)$ denote the fraction of documents that contain entities of concept C in dataset DS. We call $d(C)$ the *frequency* of C over DS. Let

$d(P)$ be the total frequency of leaf concepts in the partition of P. The fraction of the documents that contain information about entities in C is $\frac{d(C)}{d(P)}$. For example, assume that the mentions to entities of concept *Trachoma* appear more frequently in dataset DS than the ones of concept *Hordeolum*. Also, assume that we annotate only *Eye-Infections* in DS. Given query $(Hordeolum, pain)$, it is more likely for articles about Trachoma to appear in the top k answers than the ones about Hordeolum. Hence, the total contribution of partition P in improving precision and recall of answering queries with some concept $C \in \texttt{part}(P)$ is $\sum_{C \in \texttt{part}(P)} \frac{d(C)}{d(P)}$.

We call the fraction of queries in $\mathcal{Q}$ whose concept is C the *popularity* of C in $\mathcal{Q}$ and denote it by $u(C)$. Consider designs $\mathcal{S}_1$ and $\mathcal{S}_2$ over taxonomy $\mathcal{X}$. Assume that design $\mathcal{S}_1$ and $\mathcal{S}_2$ equally improve the values of precision (recall) for queries of all concepts except for $C_1, C_2 \in \mathcal{C}$. Also, let the precision (recall) of C_1 be improved more by $\mathcal{S}_1$ than by $\mathcal{S}_2$. Similarly, assume that the precision (recall) of C_2 is increased more by $\mathcal{S}_2$ than by $\mathcal{S}_1$. Given all other conditions are the same, if we have $u(C_1) > u(C_2)$, design $\mathcal{S}_1$ improves the total precision (recall) of queries in $\mathcal{Q}$ more than $\mathcal{S}_2$. Hence, we should take into account the popularities of concepts to compute the amount of improvement achieved by a design over $\mathcal{Q}$. Therefore, we modify the formula to estimate the contribution of partition P in improving precision (recall) of answering queries in $\mathcal{Q}$ as $\sum_{C \in \texttt{part}(P)} \frac{u(C)\ d(C)}{d(P)}$. Given all other conditions are the same, the larger this value is, the more likely it is that the query interface will achieve larger values of precision (recall) in the top k answers over queries in $\mathcal{Q}$.

Annotators may make mistakes in identifying the correct concepts of entities [8]. An annotator may recognize some instances concepts that are not P as the ones in P. For example, the annotator of concept *person* may identify *Lincoln*, the movie, as a person. The *accuracy* of annotating concept P over DS is the number of correct annotations of P divided by the number of all annotations of P in DS. We denote the accuracy of annotating concept P over DS as $\texttt{pr}(P)$. Hence, we refine our estimate to: $\sum_{C \in \texttt{part}(P)} \frac{u(C)\ d(C)}{d(P)}\ \texttt{pr}(P)$.

So far, we have estimated the relative improvement gained by $\mathcal{S}$ for queries whose concepts belong to some partitions in $\mathcal{S}$. Consider query $Q : (C, T)$ such that C does not belong to any partition in $\mathcal{S}$, i.e., C is a free concept. The query interface has to examine all documents in the dataset to answer Q. Thus, the fraction of returned answers for Q that contains some instance of C is $d(C)$. The more instances of C appear in the dataset DS, the more likely it is that the returned answers to Q refer to entities in C. Hence, it is more likely that they contain some relevant answers for Q. Using a similar argument as the one used for non-free concepts, the total contribution of the free concepts of design $\mathcal{S}$ is $\sum_{C \in \texttt{free}(\mathcal{S})} u(C)d(C)$. The following function estimates the relative improvement in the precision (recall) in the top k answers for all concepts.

Definition 3.2. The queriability of design $\mathcal{S}$ from taxonomy $\mathcal{X}$ over dataset DS and query workload $\mathcal{Q}$ is

$$QU(\mathcal{S}) = \sum_{P \in \mathcal{S}} \sum_{C \in \texttt{part}(P)} \frac{u(C)\ d(C)\ \texttt{pr}(P)}{d(P)} + \sum_{C \in \texttt{free}(\mathcal{S})} u(C)d(C)$$

Similar to the definition of *annotation benefit* proposed in [13], the first term in the formula of queriability reflects the amount of improvement in query answering from concepts in the design, and the second term reflects the benefit from concepts not in the design. However, the queriability also uses structural information of the taxonomies to help estimate the improvement in the effectiveness of query answering.

Similar to other optimization problems in data management, such as query optimization, the complete information about the parameters of the objective function, i.e. frequencies and popularities of concepts, may not be available at the design-time. Nevertheless, our empirical results in Section 5 and in our technical report [14] indicate that one can effectively estimate these parameters using a small sample of the full dataset. We leave more principled approaches to parameter estimation for future work.

3.4 Cost-Effective Conceptual Design Problem

We have reviewed the literature on concept annotation and information extraction and talked to the experts to build a reasonable and general cost model for concept annotation. The types of costs for creating annotated datasets vary based on the methodology used for developing concept annotators. One may categorize the available methodologies to rule-based methods and approaches based on machine-learning techniques [8]. In rule-based annotation, developers write a set of rules for each concept to detect and extract its instances in a dataset. Rule-based approach is the dominating method in commercial information extraction systems [5].

If one adapts a machine-learning approach to annotate the entities of a concept, he has to provide a set of training examples for the concept, which may be costly and time-consuming. This stage is particularly resource-intensive in specific domains, such as medicine. Researchers have proposed the idea of distant supervision to reduce the overhead of providing training data for concept extraction. However, distant supervision typically requires knowledge-bases in the domain of extraction with instances of the extracted concepts, which is not always available. One may also generate training data for a concept by coding how the concept appears in the unstructured dataset in a programming language [9]. This method, however, needs a domain expert to learn a programming language and code her knowledge in a piece of program. Furthermore, the developer has to distinguish and select relevant features for each concept. All concepts may share some general features. However, developers have to also engineer considerable number of features specific to each concept [11]. For example, an informative feature for *zip-code* is that its instances have 5 digits and a helpful feature for *person* is that its entities start with a capital letter. These features may be given to a classifier or a probabilistic

graphical model, or coded as first order logic formulas in a Markov Logic Network. After developing the concept annotator, domain experts may review and evaluate the annotation of each concept [6]. This process may repeat multiple times to generate accurate annotations of the concept. As most underlying datasets frequently evolve, the aforementioned steps have to be redone after a while for each concept in both approaches [3].

Given taxonomy $\mathcal{X} = (R, \mathcal{C}, \mathcal{R})$ and dataset DS, one may assign a real number to each concept $C \in \mathcal{C}$ that reflects the amount of resources required for the annotations of C in DS. Let function $w : \mathcal{C} \to \mathbb{R}^+$ map each concept $C \in \mathcal{C}$ to a real number that reflects the cost of annotating C in DS. The cost of annotating a dataset under design $\mathcal{S}$ is the sum of the costs of all concepts in $\mathcal{S}$. Budget B is a positive real number that represents the amount of available resources for annotating the dataset. We define Cost-Effective Conceptual Design problem (CECD) as follows.

PROBLEM 3.3. *Given taxonomy $\mathcal{X}$, dataset DS in the domain of $\mathcal{X}$, query workload $\mathcal{Q}$, and budget B, find design $\mathcal{S}$ over $\mathcal{X}$ that has the maximum queriability and $\sum_{C \in \mathcal{S}} w(C) \leq B$.*

Given a tree taxonomy such that all nodes other than its root are leaf concepts, the problem of CECD over tree taxonomies becomes the general case of CECD over a set of concepts [13]. Since the problem of CECD over a set of concepts is NP-hard [13], the problem of CECD over tree taxonomies is NP-hard. Because the algorithms in [13] do not consider the superclass/subclass relationships between concepts, they do not effectively solve CECD for tree taxonomies. Our experiments in Section 5.2 show that these algorithms find less accurate solutions to CECD over tree taxonomies than the ones that consider superclass/subclass relationships.

4 APPROXIMATION ALGORITHM

We propose an approximation algorithms called Level-wise algorithm to solve the problem of CECD using a greedy approach. It returns a design whose concepts are all from a same level of the input taxonomy. Our algorithm finds the design with maximum queriability for each level using APM algorithm proposed in [13], which finds the cost-effective subset of concepts over a set of concepts. Our algorithm eventually delivers the design with largest queriability across all levels.

Precisely, let $\mathcal{C}[i]$ be the set of all concepts of depth i in $\mathcal{X}$ For any concept $E \in \mathcal{C}[i]$ we define its popularity $u_i(E)$ to be the total popularity of its descendant leaves in $\mathcal{X}$. Level-wise algorithm calls APM to find the cost-effective subset of concepts for every $\mathcal{C}[i]$. It provides APM with the popularities and costs of concept in $\mathcal{C}[i]$. Level-wise algorithm then compares various selected designs across $\mathcal{C}[i]$s and keeps the answer with maximum queriability, denoted as $\text{sol}_{\texttt{level}}$. The algorithm also computes the queriability delivered by picking only the leaf node with maximum popularity in $\mathcal{X}$ called $\text{sol}_{\texttt{max}}$. The algorithm returns the solution with greatest queriability among $\text{sol}_{\texttt{level}}$ and $\text{sol}_{\texttt{max}}$. The APM algorithm runs in $O(|\mathcal{C}| \log |\mathcal{C}|)$ time, where $|\mathcal{C}|$ is the size of its input set of concepts. Thus, the time complexity of Level-wise algorithm is $O(h|\mathcal{C}| \log |\mathcal{C}|)$ over taxonomy $\mathcal{X} = (R, \mathcal{C}, \mathcal{R})$, where h is the number of levels in $\mathcal{X}$. If the taxonomy is not balanced, the popularities of all concepts in some level l may not sum up to 1. Hence, the algorithm does not consider leaf concepts that are not descendant of any concept in l. To resolve this problem, when running APM algorithm for level l, we consider leaf concepts that are not descendant of any concept in level l as members of l.

Taxonomy	T1	T2	T3	T4	T5	T6	T7	T8
#Concept	10	17	17	28	63	185	279	387
#Height	2	2	3	7	6	8	8	9
#Total queries	648	256	146	4219	4888	4728	5216	5259
#Documents	68k	267k	88k	1470k	1470k	1470k	1470k	1470k

Table 1: The sizes and heights of taxonomies and the sizes of their corresponding query workloads and datasets.

	budget	T1 Oracle	T1 QM	T2 Oracle	T2 QM	T3 Oracle	T3 QM
Uniform Cost	0.1	0.149	0.149	0.241	0.232	0.222	0.210
	0.2	0.168	0.168	**0.303**	0.285	**0.281**	0.269
	0.3	0.177	0.177	0.318	0.315	0.304	0.304
	0.4	0.192	0.192	0.320	0.318	0.306	0.304
	0.5	0.193	0.193	0.326	0.324	0.306	0.306
	0.6	0.195	0.195	0.326	0.326	0.306	0.306
	0.7	0.195	0.195	0.326	0.326	0.306	0.306
Random Cost	0.1	0.124	0.124	0.264	0.262	**0.248**	0.239
	0.2	0.163	0.163	0.302	0.295	0.288	0.281
	0.3	0.179	0.177	0.317	0.316	0.304	0.304
	0.4	0.187	0.183	0.323	0.318	0.306	0.306
	0.5	0.193	0.192	0.326	0.323	0.306	0.306
	0.6	0.194	0.194	0.326	0.325	0.306	0.306
	0.7	0.195	0.195	0.326	0.326	0.306	0.306

Table 2: Average p@3 for Oracle and QM. Statistically significant differences between Oracle and QM are marked in bold.

Sometimes, it may be easier to use and manage designs whose concepts are not subclass/superclass of each other. We call such a design a *disjoint design*. One may restrict the feasible solutions in the CECD problem to disjoint designs. We call this case of CECD, *disjoint* CECD. Recent empirical results suggest that the distribution of concept frequencies over a large collection generally follows a *power law* distribution [15]. We show that the Level-wise algorithm has a bounded and reasonably small worst-case approximation ratio for CECD with disjoint solution given that the distribution of concept frequencies follows a power law distribution.

THEOREM 4.1. *Let $\mathcal{X} = (R, \mathcal{C}, \mathcal{R})$ be a taxonomy with height h and the minimum accuracy of $\texttt{pr}_{\min} = \min_{C \in \mathcal{C}} \texttt{pr}(C)$. The* Level-wise *algorithm is a $O(\frac{h + \log |\mathcal{C}|}{\texttt{pr}_{\min}})$-approximation for the* CECD *problem with disjoint solution on $\mathcal{X}$ and budget B given that the distribution of frequencies in $\mathcal{C}$ follows a power law distribution.*

Because concept annotation algorithms are reasonably accurate, $\texttt{pr}_{\min}$ is generally large [8].

5 EXPERIMENTS

Taxonomies and dataset: We have extracted 8 tree taxonomies from YAGO ontology version 2008-w40-2 [12] to validate our model and evaluate effectiveness and efficiency of our proposed algorithm. YAGO organizes its concepts using subclass relationships in a DAG with a single root.

		T1		T2		T3		T4		T5		T6		T7		T8	
	budget	APM	LW	APM	LW	APM	LW	APM	LW	APM	LW	APM	LW	APM	LW	APM	LW
Uniform Cost	0.1	0.089	**0.103**	0.234	0.232	0.208	0.210	0.158	**0.179**	0.178	**0.206**	0.229	**0.240**	0.243	**0.254**	0.250	**0.259**
	0.2	0.149	**0.164**	0.253	0.285	0.258	0.269	0.177	**0.212**	0.214	**0.227**	0.244	**0.248**	0.259	**0.261**	0.262	0.263
	0.3	0.164	0.164	0.292	**0.316**	0.288	0.297	0.191	**0.231**	0.228	**0.242**	0.247	0.248	0.260	0.261	0.262	0.263
	0.4	0.164	**0.183**	0.320	0.318	0.297	**0.304**	0.215	**0.240**	0.237	**0.248**	0.248	0.248	0.261	0.261	0.263	0.263
	0.5	0.183	**0.192**	0.323	0.323	0.304	0.306	0.229	**0.241**	0.241	**0.250**	0.248	0.248	0.261	0.261	0.263	0.263
	0.6	0.192	0.194	0.323	0.326	0.304	0.306	0.229	**0.241**	0.249	0.250	0.248	0.248	0.261	0.261	0.263	0.263
	0.7	0.193	0.195	0.323	0.326	0.306	0.306	0.235	0.241	0.249	0.250	0.248	0.248	0.261	0.261	0.263	0.263
	0.8	0.195	0.195	0.323	0.326	0.306	0.306	0.239	0.241	0.249	0.250	0.248	0.248	0.261	0.261	0.263	0.263
	0.9	0.195	0.195	0.323	0.326	0.306	0.306	0.241	0.241	0.250	0.250	0.248	0.248	0.261	0.261	0.263	0.263
Random Cost	0.1	0.097	0.104	0.239	0.257	0.240	0.235	0.177	**0.189**	0.183	**0.210**	0.231	**0.240**	0.245	**0.256**	0.255	**0.259**
	0.2	0.111	0.145	0.263	**0.291**	0.263	0.283	0.180	**0.212**	0.216	**0.230**	0.245	**0.248**	0.259	0.261	0.262	0.263
	0.3	0.122	**0.175**	0.300	**0.317**	0.275	**0.301**	0.188	**0.230**	0.231	**0.242**	0.247	0.248	0.260	0.261	0.263	0.263
	0.4	0.164	**0.185**	0.321	0.320	0.294	**0.305**	0.212	**0.239**	0.239	**0.248**	0.248	0.248	0.261	0.261	0.263	0.263
	0.5	0.185	**0.192**	0.323	0.322	0.301	**0.306**	0.223	**0.241**	0.245	0.250	0.248	0.248	0.261	0.261	0.263	0.263
	0.6	0.192	0.195	0.323	0.325	0.304	0.306	0.228	0.241	0.248	0.250	0.248	0.248	0.261	0.261	0.263	0.263
	0.7	0.194	0.195	0.325	0.325	0.306	0.306	0.235	0.241	0.249	0.250	0.248	0.248	0.261	0.261	0.263	0.263
	0.8	0.195	0.195	0.325	0.326	0.306	0.306	0.239	0.241	0.249	0.250	0.248	0.248	0.261	0.261	0.263	0.263
	0.9	0.195	0.195	0.326	0.326	0.306	0.306	0.240	0.241	0.250	0.250	0.248	0.248	0.261	0.261	0.263	0.263

Table 3: Average p@3 for APM and LW over T1-T8. Statistically significant differences between APM and LW are marked in bold.

We have created the breath-first tree of YAGO and selected the concepts to create the taxonomies. We have used the collection of English Wikipedia articles from the Wikipedia dump of October 8, 2008 that is annotated by concepts from YAGO ontology in our experiments [12]. For each taxonomy in our sets of taxonomies, we have extracted a subset of the original Wikipedia collection where each document contains at least a mention to an entity of a concept in the taxonomy. We use each dataset in the experiments over its corresponding taxonomy. Table 1 shows the properties of the taxonomies and their corresponding dataset.

Query Workload: We use a subset of Bing query log whose target URLs, i.e., relevant answers, are Wikipedia articles. Because the query log does not have the concepts behind its queries, we adapt an automatic approach to find the most specific concept associated with each query. We label each query by the concept of the matching instance in its relevant answer(s). Using this method, we create a query workload per each of our datasets. Table 1 shows the information about our query workloads. We use two-fold cross validation to calculate the popularities, u, of concepts in each taxonomy over their corresponding query workload.

Query Interface: We index our datasets using Lucene (*lucene. apache.org*). Given a query, we rank its candidate answers using BM25 ranking formula [7]. Then, we apply the information about the concepts in the query and documents to return the answers whose matching instances have the same concept as the concept of the query. If the concept in the query has not been annotated from the collection, the query interface returns the list of document ranked by BM25 method without any modification. We have implemented our query interface and algorithms in Java 1.7 and performed our experiments on a Linux server with 64 GB of main memory and two quad core processors.

Effectiveness Metric: Because all queries in our query workloads have one or two relevant answers, we measure the ranking quality of answering queries over a dataset using precision at top 3 answers (p@3) [7]. One may also use larger number of top answers to measure precision and ranking quality of the results. However, since the total number of relevant answers in our query workloads are less than three, the maximum possible values for p@k, and consequently the reported values, will be very small. Thus, using larger values of k makes comparing and analyzing the effectiveness of query results rather hard. We have reported the results using other effectiveness metrics, such as MRR and recall, at [14]. We measure the statistical significance of our results using the paired-t-test at a significant level of 0.05.

Cost Models: We use two models for generating costs of concept annotation. First, we assign a randomly generated cost to each concept in a taxonomy. The results reported for this model are averaged over 5 sets of random cost assignments per budget. We call this model *random cost* model. If there is not any reliable estimation available for the cost of annotating concepts, an organization may assume that all concepts are equally costly. Hence, in our second cost model, we assume that all concepts in the input taxonomy have equal cost. We name this model *uniform cost* model. We use a range of budgets between 0 and 1 with a step size of 0.1 where 1 means sufficient budget to annotate all concepts in a taxonomy.

5.1 Validating Queriability Function

Oracle: Given a fixed budget, Oracle enumerates all feasible designs over the taxonomy, and picks the design with maximum value of average p@3.

Queriability Maximization (*QM*): QM enumerates all feasible designs over the taxonomy and returns the one with the maximum queriability as computed in Section 3.3.

As these algorithms enumerate all feasible designs, it is not possible to run them over large taxonomies. Hence, we run these algorithms over small taxonomies: T1, T2, and T3. Table 2 shows the average p@3 of Oracle and QM over T1, T2, T3 under uniform cost model and random cost model. We do not show the values of average p@3 for budgets greater than 0.7 as they are equal to the average p@3 for budgets 0.7. Generally, the designs picked by QM deliver close p@3 values to the ones selected by Oracle.

5.2 Effectiveness of the Proposed Algorithm

We compare the level-wise (LW) algorithm with the *APM* algorithm from [13] that finds a design over a set of concepts. We use all concepts in the taxonomy as a set for an input to APM and set $\epsilon = 0.01$ for both APM and LW. Table 3 shows the values of average p@3 for APM and LW over all taxonomies and cost models. Overall, the designs returned by LW improve the effectiveness of answering queries for all taxonomies more than the ones returned by APM. As APM does not consider the structural information of the taxonomy, it often picks popular concepts that are ancestor or descendant of each other. LW uses structural information of the taxonomy and avoid this problem. Nevertheless, we observe less significant improvement of LW over APM for larger taxonomies such as T6, T7 and T8. Due to very skewed distributions of concept popularity in T6, T7 and T8, budget of 0.3 is usually sufficient to create a design that includes all concepts which appear in the query workloads. Hence, LW generally performs significantly better than APM for budget less than 0.3 for T6, T7 and T8. For larger budgets, both APM and LW return the designs with the same average p@3.

5.3 Efficiency of the Proposed Algorithm

We measure the running times of LW over moderate and large taxonomies, i.e., T4, T5, T6, T7 and T8. The average running times of LW for T4, T5, T6, T7 and T8 are 2, 2, 5, 6 and 6 seconds per budget, respectively. Thus, LW is efficient for a design time task.

6 CONCLUSION

We introduced the problem of cost-effective conceptual design using taxonomies. We proposed an efficient approximation (LW) algorithm for the problem over tree taxonomies. Our empirical results show that LW is generally effective and scalable.

ACKNOWLEDGMENTS

This work is supported by NSF grant number 1421247 and IIS-1423238.

REFERENCES

[1] Serge Abiteboul, Ioana Manolescu, Philippe Rigaux, Marie Rousset, and Pierre Senellart. 2011. *Web Data Management*. Cambridge University Press.
[2] Michael Anderson and others. 2013. Brainwash: A Data System for Feature Engineering. In *CIDR*.
[3] Pankaj Gulhane and others. 2011. Web-scale Information Extraction with Vertex. In *ICDE*.
[4] Pallika Kanani and others. 2012. Selecting Actions for Resource-bounded Information Extraction using Reinforcement Learning. In *WSDM*.
[5] Benny Kimelfeld. 2014. Database Principles in Information Extraction. In *PODS*.
[6] Ke Liu and others. 2015. MeSHLabeler: improving the accuracy of large-scale MeSH indexing by integrating diverse evidence. *Bioinformatics* 31, 13 (2015).
[7] Christopher Manning and others. 2008. *An Introduction to Information Retrieval*. Cambridge University Press.
[8] Andrew McCallum. 2005. Information Extraction: Distilling Structured Data From Unstructured Text. *ACM Queue* (2005), 48–57.
[9] Alexander Ratner and others. 2016. Data Programming: Creating Large Training Sets, Quickly. *arXiv:1605.07723* (2016).
[10] M. Sanderson. 2008. Ambiguous Queries: Test Collections Need More Sense. In *SIGIR*.
[11] Sandeepkumar Satpal, Sahely Bhadra, Sundararajan Sellamanickam, Rajeev Rastogi, and Prithviraj Sen. 2011. Web information extraction using markov logic networks. In *KDD*.
[12] Fabian Suchanek and others. 2007. YAGO: A Core of Semantic Knowledge Unifying WordNet and Wikipedia. In *WWW*.
[13] Arash Termehchy, Ali Vakilian, Yodsawalai Chodpathumwan, and Marianne Winslett. 2014. Which Concepts Are Worth Extracting?. In *SIGMOD*.
[14] Ali Vakilian and others. 2016. Cost-Effective Conceptual Design Using Taxonomies. *arXiv:1503.05656* (2016).
[15] Wentao Wu, Hongsong Li, Haixun Wang, and Kenny Zhu. 2012. Probase: A Probabilistic Taxonomy for Text Understanding. In *SIGMOD*.