Noname manuscript No.
(will be inserted by the editor)

Cost-Effective Conceptual Design Using Taxonomies

Yodsawalai Chodpathumwan · Ali Vakilian · Arash Termehchy · Amir Nayyeri

Received: date / Accepted: date

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 efficient approximation and pseudo-polynomial time algorithms for several cases of the problem. Our extensive empirical studies validate our framework and show accuracy and efficiency of our algorithms.

Keywords concept annotation; information extraction; data curation; taxonomies; cost-effective data curation

The first two authors have equally contributed to the paper.

Y. Chodpathumwan
University of Illinois at Urbana-Champaign, Urbana, IL,
E-mail: ychodpa2@illinois.edu

A. Vakilian
Massachusetts Institute of Technology, Cambridge, MA,
E-mail: vakilian@mit.edu

A. Termehchy
Oregon State University, Corvallis, OR,
E-mail: termehca@oregonstate.edu

A. Nayyeri
Oregon State University, Corvallis, OR,
E-mail: nayyeria@oregonstate.edu

```
<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>
```

Fig. 1 Medical article excerpts

1 Introduction

Taxonomies provide shared understandings of concepts in domains of interests [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 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 2 depicts fragments of the Medical Subject Heading (MeSH) taxonomy, in which nodes denote concepts and edges show subclass relationships [39]. Figure 3 shows the medical article excerpts in Figure 1 whose entities are annotated by their concepts from MeSH taxonomy. Now, our user

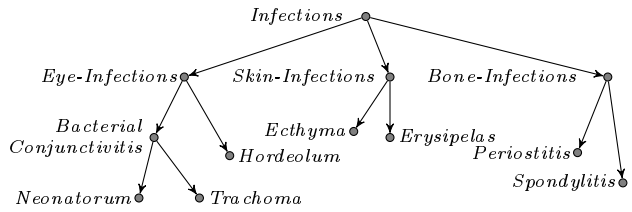

Fig. 2 Fragments of MeSH taxonomy

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

Fig. 3 Annotated medical article excerpts

may mention the concept *Trachoma* in her query and 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 MeSH taxonomy [34, 39]. Researchers have used the ProBase taxonomy to extract concepts from Web data [27]. The NIH funded Gene Ontology Consortium (*geneontology.org*) encourages researchers to annotate their datasets using a standard taxonomy so their datasets become more accessible to other researchers. We have been recently asked by some botanists to annotate a collection on plant biology by the concepts in Plant Ontology (*plantontology.org*). Organizations also use taxonomies to extract entities in general domains. For example, researchers have used *ProBase* taxonomy to extract concepts in general domains from Web data [27, 55]. 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 all relevant concepts in a given taxonomy from a data set to answer all queries effectively. However, an organization has to spend significant amounts of time, financial and computational resources, and manual labor to accurately extract entities of a concept in a large data set [3, 11, 24, 28, 30, 31, 48, 50]. The organization usually has to develop or obtain a complex program called a *concept annotator* to annotate instances of a concept from a

```
<article>
 <Eye-Infections>Granular conjunctivitis</Eye-
 Infections> causes pain in 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>
```

Fig. 4 Medical article excerpts annotated with more general concepts

collection of documents [37]. It is not uncommon for an annotator to have thousands of manually written programming rules, which takes a great deal of resources to write and debug [37]. One may also use machine learning algorithms to develop an extractor for a concept to avoid hand-tuned programming rules. In this approach, developers have to find a set of *relevant features* for the learning algorithm. As the specifications of relevant features *not* usually clear, developers have to find the relevant features through trial and error over numerous iterations, which takes a great deal of time and effort [2, 3]. Moreover, developers have to also create training data, which require additional time and manual labor. It is more resource-intensive to develop annotators for concepts in specific domains, such as biology, as it requires expensive communication between domain experts and developers. Current studies indicate that these communications are not often successful and developers themselves have to go through the data to find the relevant features in these domains [3]. As most concept annotators perform complex text analysis, it may take them days to process a large dataset and produce an annotated collection [28, 31, 48]. Since concept annotators may not be sometimes sufficiently accurate, domain experts have to review and revise the results of the annotations [34]. It is estimated that annotating each article in MEDLINE/PubMED collection using concepts in MeSH taxonomy costs about $9.4 [34].

Because the structure and content of underlying datasets evolve over time, annotators should be regularly rewritten and repaired. Many annotators need to be rewritten in average almost every two months [24]. Recent studies show that many concept annotators need to be rewritten in average about every two months [24]. Thus, enterprises often have to repeat the resource-intensive steps of developing a concept annotator to maintain an up-to-date annotated data set.

Because the financial or computational resources of an organization are limited, it may not be able to afford

to develop and maintain annotators for all concepts in a taxonomy. Also, many users may need an annotated data set quickly and cannot wait days for an (updated) annotated collection [48]. For example, a reporter who pursues some breaking news and an epidemiologist that follows the pattern of a new potential pandemic on the Web and social media need relevant answers to their queries fast. They may not want to wait for organizations to (re-)write and (re-)deploy the annotators for all concepts in their domains of interests. Hence, an organization may be able to 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 enterprise 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 3. 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 use their intuitions to discover cost-effective conceptual designs from taxonomies. Because most taxonomies contain hundreds of concepts [49], this approach does not scale for real-world applications. We call this problem 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 2. We also formally define the CECD problem over tree-shaped taxonomies in Section 2.
- As the CECD problem is NP-hard, we propose an efficient approximation algorithm for it in Section 3. We also propose an exact algorithm for the problem with pseudo-polynomial running time in Section 4.
- Annotating the ancestor(s) of a concept in a tree taxonomy may reduce the cost of extracting that concept. We propose an exact pseudo-polynomial time algorithm for CECD for this case in Section 5.

Table 1 Summary of notions and symbols.

Notions/Symbols	Definitions
R	root concept (of a taxonomy)
$\mathcal{C}$	finite set of concepts
$\mathcal{R}$	set of subclass relations of concepts in $\mathcal{C}$: $(C, D) \in \mathcal{R}$ iff D is a subclass of C
$\mathcal{X}$	taxonomy $\mathcal{X} = (R, \mathcal{C}, \mathcal{R})$
$\mathcal{Q}$	query workload
$Q : (C, T)$	query with concept C and set of terms T
B	budget
design $\mathcal{S}$ over $\mathcal{X}$	non-empty subset of $\mathcal{C} \setminus \{R\}$
$\texttt{part}(C)$	partition of concept C (refer to Definition 1)
$\texttt{part}(\mathcal{S})$	set of $\texttt{part}(C)$ for all $C \in \mathcal{S}$
$\texttt{free}(\mathcal{S})$	subset of leaf concepts in $\mathcal{X}$ that do not belong in any partition in $\texttt{part}(\mathcal{S})$
$d(C)$	*frequency* of C: a fraction of documents in a dataset that contain entities of concept C
$u(C)$	*popularity* of C: a fraction of queries whose concepts is C
$w(C)$	*cost* of annotation of C: $w : \mathcal{C} \to \mathbb{R}^+$
$QU(\mathcal{S})$	queriability of design $\mathcal{S}$ (refer to Definition 2)

- We investigate the problem of CECD for queries that refer to multiple concepts over an unorganized set of concepts and show that it has an efficient approximation algorithm in Section 6. We extend this algorithm to solve CECD for queries with multiple concepts over tree taxonomies.
- We explore CECD over taxonomies that are directed acyclic graphs and prove that given generally accepted hypotheses, the problem does not have any approximation algorithm with reasonably small approximation ratio. We show that these results hold even for some restricted cases of the problem, such as the case where all concepts are equally costly.
- We evaluate the accuracy of our framework and effectiveness of efficiency of our algorithms using a large real-world dataset, real-world taxonomies, and samples of a real-world query workload in Section 8.

2 Cost-Effective Conceptual Design

In this section, we formally define the problem of cost-effective conceptual design over taxonomies. Table 1 contains a summary of notions and symbols that are defined in this section and used throughout the paper.

2.1 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. We identify each concept with a unique name. Some examples of concepts are *person* and *country*. An entity of concept *person* is *Albert Einstein* and an entity of concept *country* is *Jordan*. Concept C is a *subclass* of concept D if and only if we have $C \subset D$.

In this case, we call D a *superclass* of C. For example, *person* is a superclass of *scientist*. If an entity belongs to a concept C, it will belong to all its superclass's.

A taxonomy organizes concepts in a domain of interest [1]. We first investigate the properties of tree-shaped taxonomies and later in Section 7, we will explore the taxonomies that are directed acyclic graphs. Formally, we define *taxonomy* $\mathcal{X} = (R, \mathcal{C}, \mathcal{R})$ as a rooted tree, with a *root concept* R, a vertex set $\mathcal{C}$ and an edge set $\mathcal{R}$. $\mathcal{C}$ is a finite set of concepts. For $C, D \in \mathcal{C}$, we have $(C, D) \in \mathcal{R}$ if and only if 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, concepts *Trachoma* and *Hordeolum* are leaf concepts in Figure 2. Let $ch(C)$ denote the children of concept C. For the sake of simplicity, we assume that $\cup_{D \in ch(C)} D = C$ for all concepts C in a taxonomy.

Each dataset is a set of documents. Dataset DS is in the domain of taxonomy $\mathcal{X}$ if and only if some entities of concepts in $\mathcal{X}$ appear in some documents in DS. For instance, the set of documents in Figure 1 is in the domain of the taxonomy shown in Figure 2. An entity in $\mathcal{X}$ may appear in several documents 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. Some example queries are *(person, {Michael Jordan})* or *(location,{Jordan})*. This type of queries has been widely used to search and explore annotated datasets [8, 13, 41]. Empirical studies on real world query logs indicate that the majority of entity centric queries refer to a single entity [44]. We first consider queries that refer to a single entity and extend our model to support queries with multiple concepts in Section 6.

2.2 Conceptual Design

A *conceptual design* $\mathcal{S}$ over taxonomy $\mathcal{X} = (R, \mathcal{C}, \mathcal{R})$ is a non-empty subset of $\mathcal{C} \setminus \{R\}$. For brevity, we refer to conceptual design as *design*. A design divides the set of leaf nodes in $\mathcal{C}$ into some partitions. As shown in Section 1, annotating a concept may help answering queries whose entities belong to the descendants of that concept. Consider again the fragment of MeSH taxonomy shown in Figure 2. Assume a user wants to find information about the types of pain associated with Trachoma over a dataset and submits query *(Trachoma, {pain})* to the query interface. Suppose we have annotated the dataset with the concept in the design {*Eye-Infections*}. Because the articles about *Trachoma* are within the ones annotated by *Eye-Infections*, the query interface may return only articles annotated with *Eye-Infections* to the user. This annotation helps the query interface not to return the non-relevant articles about skin or bone infections. We say that *Trachoma* is in the *partition* of *Eye-Infections*. Now, suppose we annotate the dataset with the concepts in the design {*Bacterial Conjunctivitis*, *Eye-Infections*}. The articles annotated by *Eye-Infections* and *not* annotated by *Bacterial Conjunctivitis* do *not* contain any information about the concept *Trachoma*. Hence, the query interface may return only articles annotated with *Bacterial Conjunctivitis* in the response of the query *(Trachoma, {pain})*. Due to the annotation of *Bacterial Conjunctivitis*, annotating *Eye-Infections* does not help answering queries with concept *Trachoma*. Thus, *Trachoma* is in the partition of *Bacterial Conjunctivitis* for this design. We formally define the *partition* of a concept in a design as follows.

Definition 1 Let $\mathcal{S}$ be a design over taxonomy $\mathcal{X} = (R, \mathcal{C}, \mathcal{R})$, and let $C \in \mathcal{S}$. The **partition** of C, denoted as $\texttt{part}(C)$, is a maximal subset of leaf nodes in $\mathcal{C}$ such that, for all $D \in \texttt{part}(C)$, we have either $D = C$ or C is the lowest ancestor of D in $\mathcal{S}$.

We denote a set of all partitions induced by all concepts in a design $\mathcal{S}$ as $\texttt{part}(\mathcal{S})$.

Example 1 Consider the taxonomy described in Figure 5. Let the design $\mathcal{S}_1$ be {*Eye-Infections, Skin-Infections*}. The lowest ancestor in $\mathcal{S}_1$ of *Neonatorum*, *Trachoma* and *Hordeolum* is *Eye-Infections*, and the lowest ancestor in $\mathcal{S}_1$ of *Ecthyma* and *Erysipelas* is *Skin-Infections*. Hence, we have that $\texttt{part}(\textit{Eye-Infections}) = \{\textit{Neonatorum}, \textit{Trachoma}, \textit{Hordeolum}\}$ and $\texttt{part}(\textit{Skin-Infection}) = \{\textit{Ecthyma}, \textit{Erysipelas}\}$.

Example 2 Consider the taxonomy described in Figure 6. Let the design $\mathcal{S}_2$ be {*Eye-Infections, Bacterial Conjunctivitis*}. The lowest ancestor in $\mathcal{S}_2$ of *Neonatorum* and *Trachoma* is *Bacterial Conjunctivitis*, and the lowest ancestor in $\mathcal{S}_2$ of *Hordeolum* is *Eye-Infections*. Hence, we have that $\texttt{part}(\textit{Bacterial Conjunctivitis}) = \{\textit{Neonatorum}, \textit{Trachoma}\}$ and $\texttt{part}(\textit{Eye-Infections}) = \{\textit{Hordelum}\}$.

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 any concept in $\mathcal{S}$.

Example 3 Consider the design $\mathcal{S}_1 = \{\textit{Eye-Infections}, \textit{Skin-Infections}\}$ over the taxonomy described in Figure 5. The free concepts of $\mathcal{S}_1$, $\texttt{free}(\mathcal{S}_1)$, are {*Periostitis*, *Spondylitis*} as they do not belong to any partition of $\mathcal{S}_1$.

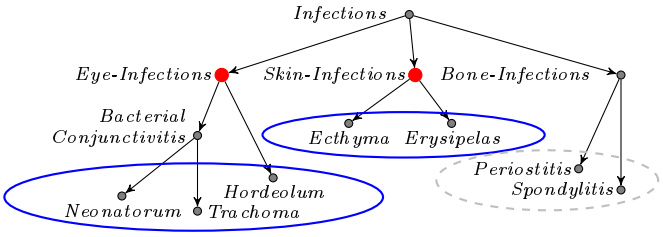

Fig. 5 The concepts in red, *Eye-Infections* and *Skin-Infections*, denote the design. The blue curves denote the partitions created after annotating the design, and the dashed curved shows the free concepts of the selected design.

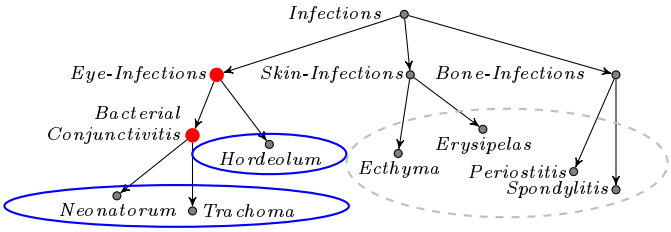

Fig. 6 The concepts in red, *Eye-Infections* and *Bacterial Conjunctivitis*, denote the design. The blue curves denote the partitions created after annotating the design, and the dashed curved shows the free concepts of the selected design.

Example 4 Consider the design $\mathcal{S}_2$ = {*Eye-Infections*, *Infections*} over the taxonomy described in Figure 6. The free concepts of $\mathcal{S}_2$, $\texttt{free}(\mathcal{S}_2)$, are {*Spondylitis*, *Periostitis*, *Ecthyma*, *Erysipelas*}.

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 if and only if for each concept $C \in \mathcal{S}$, 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-Infections, Skin-Infections*} over the taxonomy in Figure 2. 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-Infections*, are annotated by *Eye-Infections* and all instances of concepts *Ecthyma* and *Erysipelas* that are in the partition of *Skin-Infections*, are annotated by *Skin-Infections* in the dataset.

Intuitively, different designs may improve the effectiveness of answering queries differently. Consider a dataset in the domain of the taxonomy in Figure 5 in which almost all queries seek information about skin infections. If the query interface uses the design {*Eye-Infections, Skin-Infections*}, it can process queries that are about skin infection only over the documents that contain information about skin infections. But, if the query interface uses the design {*Eye-Infections, Bacterial Conjunctivitis*}, it has to process queries about skin infections over all documents in the dataset; many of which do not contain any information about skin infections. Hence, the query interface may return more non-relevant answers for most queries than the case where it uses {*Eye-Infections, Skin-Infections*}. Moreover, as explained in Section 1, a design with more specific concepts, e.g., leafs in the taxonomy, helps the query interface to pinpoint the relevant documents more effectively than the designs with more general concepts. Because annotating documents and instances of different concepts may take different amounts of time and/or financial and computational resources, each design may have a distinct cost. Hence, finding the most effective design becomes an optimization problem that seeks the design that improves the effectiveness of answering queries the most and satisfies certain cost constraints. To formalize this problem, we first precisely quantify the amount by which a design improves the effectiveness of answering queries in Section 2.3. Then, in Section 2.4, we present a cost model for building and maintain annotations for a design and formally state the problem of finding the most effective design.

2.3 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 the returned list of answers is usually measured using standard metrics of *precision* and *recall* [35]. The precision of the returned list of answers is the fraction of relevant answers for Q in the returned 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. It has been shown that most information needs over annotated data sources are precision-oriented [13, 16]. Hence, we measure the effectiveness of the returned answers using precision-oriented metrics. More precisely, we use the standard metric of precision at k ($p@k$ for short), which is the precision of the top-k answers, to measure the ranking quality of the query result [35]. 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 the precision 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 [35]. 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. More precisely, $d(C)$ is the number of documents that contain entities of C in DS divided by the total number of documents in DS. We call $d(C)$ the *frequency* of C over DS. Let $d(P)$ be the total frequency of concepts in the partition of P, i.e., $d(P) = \sum_{C \in \texttt{part}(P)} d(C)$. The fraction of the documents that contain information about entities in C amongst those that contain information of concept P 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. That is, for each concept $C \in \texttt{part}(P)$, the contribution of C in improving the precision of answering queries with concepts C is $\frac{d(C)}{d(P)}$. Hence, the total contribution of partition P in improving the precision of answering queries 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 $\mathcal{Q}$. Let $u_{\mathcal{Q}}$ be the function that maps concept C to its popularity in $\mathcal{Q}$. When $\mathcal{Q}$ is clear from the context, we simply use u instead of $u_{\mathcal{Q}}$. Assume that design $\mathcal{S}_1$ and $\mathcal{S}_2$ equally improve the values of precision for queries of all concepts except for $C_1, C_2 \in \mathcal{C}$. Also, let the precision of C_1 be improved more by $\mathcal{S}_1$ than by $\mathcal{S}_2$. Similarly, assume that the precision 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 $u(C_1) > u(C_2)$, we have that $\mathcal{S}_1$ improves the total precision 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 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 a larger precision in top k answers over queries in $\mathcal{Q}$.

Annotators may make mistakes in identifying the correct concepts of entities in a collection [37]. An annotator may recognize some instances of concepts that are not P as 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 $\texttt{pr}_{DS}(P)$. When DS is clear from the context, we show $\texttt{pr}_{DS}(P)$ 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 $\sum_{C \in \texttt{free}(\mathcal{S})} u(C)d(C)$. We define the function that estimates the relative improvement in the value of precision in the top k answers for all concepts as follows.

Definition 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 other optimization problems in data management, such as query optimization [23], 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 8 indicate that one can effectively estimate these parameters using a small sample of the full dataset. For instance, we show that the frequencies of concepts over a dataset of more than a million documents can be effectively estimated using a sample of about four hundred documents. A more principled approach to parameter estimation is an interesting subject for future work.

2.4 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 [9, 11, 15, 15, 17, 18, 33, 37, 45]. In rule-based annotation, developers write a set of rules for each concept to detect and extract its instances in a dataset [10, 21, 47]. Rule-based approach is the dominating method in commercial information and entity extraction systems [12, 25]. This is mainly attributed to the fact that rules are effective, interpretable, and easier to customize by non-experts than the methods based on machine learning [25]. Furthermore, rules-based systems perform better than state-of-the-art machine learning methods in some specialized domains [26, 40].

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 [25]. 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 [38]. However, distant supervision typically requires knowledge-bases in the domain of extraction with instances of the extracted concepts which is not always available, particularly in a specific domain such as biology. One may also generate training data for a concept by coding how the concept appears in the unstructured dataset in a programming language [42]. 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 because many or all concepts may share some general features. For example, concept annotators may use the surrounding words of entities in text documents as features for all concepts. However, developers have to also engineer considerable number of features specific to each concept [29, 46]. 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 [29] or a probabilistic graphical model [54] or coded as first order logic formulas in a Markov Logic Network [46]. After developing the concept annotator, domain experts may review and evaluate the annotation of each concept [34, 39]. 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 [22, 24].

Hence, 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 to annotate and maintain the annotations of C in DS. Let function $w : \mathcal{C} \rightarrow \mathbb{R}^+$ map each concept $C \in \mathcal{C}$ to a real number that reflects the cost of annotating C in DS. Developers also create and maintain some preprocessing modules to tokenize the input documents and separate the (potential) named entities from other tokens, e.g., adjectives, in both rule-based method and the methods based on machine learning techniques. This cost is generally independent of the number of extracted concepts and can be viewed as a fixed cost for annotating a dataset. This model assumes that annotating certain concepts does not affect the cost and accuracies of annotating other concepts. We later relax this restriction in Section 5 and consider some dependencies between the costs of concepts in the taxonomy. Nevertheless, it usually takes significant amount of resources to develop and maintain a concept annotator even after pairing it with other annotators. Thus, it is still of interest to solve the problem where there is no dependency between costs of different concepts. The organization may predict the costs of development, deployment and maintenance of annotation programs using available methods for predicting costs of software development and maintenance [7].

The cost of annotating a dataset under design $\mathcal{S}$ is the sum of the costs the 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 1 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 $\sum_{C \in \mathcal{S}} w(C) \leq B$.

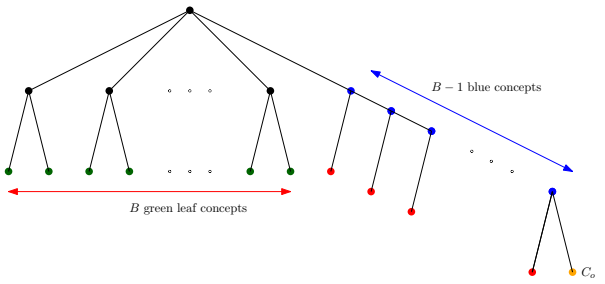

Fig. 7 An example to analyze algorithms that ignore the tree structure.

Unfortunately, the CECD problem cannot be solved in polynomial time in terms of input size unless **P** = **NP**.

Theorem 1 *The problem of* CECD *is* **NP***-hard.*

Because the approximation algorithms proposed in [50] do not consider the superclass/subclass relationships between concepts, they do not effectively solve the CECD problem for tree taxonomies. In particular, these algorithms generally choose designs with more popular concepts, i.e., concepts with larger $u(.)$ values. Because each node has generally more $u(.)$ value than its descendant concepts in the input taxonomy, these algorithms spend the budget on picking concepts in higher levels, while it may worth including concepts in lower levels of the taxonomy in the design. Our empirical studies in Section 8.3 confirm that these algorithms do not generally find accurate solutions to CECD over tree taxonomies. We also show that the algorithms in [50] have a worst-case approximation ratio of $O(|\mathcal{C}|)$ for the problem, which is a significantly inaccurate approximation even for taxonomies with a modest number of concepts, as follows.

Consider the taxonomy shown in Figure 7 where each green leaf concept C_g has $w(C_g) = 1$, $u(C_g) = (1+\epsilon)v$ and $d(C_g) = p$. For each red leaf concept C_r, $w(C_r) = B+1$, $u(C_r) = Bv$ and $d(C_r) = \frac{1}{B^2}p$. For the orange leaf concept C_0, $w(C_o) = 1$, $u(C_o) = v$ and $d(C_o) = p$. v and p are constants such that $\sum_{C\in\mathcal{C}} u(C) = \sum_{C\in\mathcal{C}} d(C) = 1$. For each non-root black concept C, $w(C) = B$ and for each blue concept C_b, $w(C_b) = 1$. Assume that the total available budget is B. The optimal solution of any algorithm that ignores the tree structure and works only with leaf concepts, is to pick exactly the green leaf nodes, which delivers the queriability of almost $B(1+\epsilon)v$. Now, assume that an algorithm considers both leaf and non-leaf concepts but not their relationships. For instance, the approximate popularity maximization (APM) algorithm in [50] selects concepts C with the largest $\frac{u(C)}{w(C)}$ values. APM picks exactly B green leaf concepts in our example and returns queriability of almost $B(1+\epsilon)v$. However, we can achieve the queriability of almost B^2v by picking all blue and orange concepts. The gap between optimal solution and any solution returned by an APM algorithm is B, which is $\frac{1}{3.5}|\mathcal{C}|$. It is an ineffective solution for taxonomies with modest numbers of concepts.

input : a taxonomy $\mathcal{X} = (R, \mathcal{C}, \mathcal{R})$
output: a conceptual design

```
1  sol_level ← 0 and sol_max ← 0;
2  for i = 0 to h do
3  |   for each concept C in distance i from the root do
4  |   |   C[i] ← C[i] ∪ {C};
5  |   end
6  |   for each leaf concept C in distance less than i
   |    from the root do
7  |   |   C[i] ← C[i] ∪ {C};
8  |   end
9  |   sol_i ← approximate solution over C[i];
10 |   sol_level ← max(sol_level, sol_i);
11 end
12 C_max ← the leaf concept with the largest u value;
13 sol_max ← u(C_max) + Σ_{C∈free(C_max)} u(C)d(C);
14 return the best of sol_level and sol_max;
```

Algorithm 1: LEVEL-WISE algorithm.

3 Approximation Algorithm

We propose an approximation algorithm 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 the APM algorithm proposed in [50], which find the cost-effective subset of concepts over a set of concepts. Our algorithm eventually delivers the design with largest queriability across all levels in the taxonomy. Algorithm 1 illustrates the LEVEL-WISE algorithm.

Precisely, let $\mathcal{C}[i]$ be the set of all concepts of level i in $\mathcal{X} = (R, \mathcal{C}, \mathcal{R})$. 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 the APM algorithm 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 largest queriability amongst $\text{sol}_{\texttt{level}}$ and $\text{sol}_{\texttt{max}}$. The APM algorithm runs in $O(|\mathcal{C}| \log |\mathcal{C}|)$, where $|\mathcal{C}|$ is the size of its input set of concepts. Thus, the time complexity of the 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 of level i may not sum up to 1. Hence, the algorithm does not consider leaf concepts that are not descendant of any concept in $\mathcal{C}[i]$. To resolve this problem, when running APM algorithm over $\mathcal{C}[i]$, we consider leaf concepts that are not descendant of any concept of level i as members of $\mathcal{C}[i]$ (lines 6-8 in Algorithm 1).

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. Our empirical results in Section 8 show that this strategy returns effective designs when the available budget is relatively small. 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 [55]. 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.

Lemma 1 *Let $C_{\max}$ be the leaf concept in taxonomy $\mathcal{X} = (R, \mathcal{C}, \mathcal{R})$ with maximum u value and let assume that distribution of u over leaf concepts follows a* power law *distribution. For every schema $\mathcal{S}$ of $\mathcal{X}$, $QU(\texttt{free}(\mathcal{S}))$ $\leq 2u(\mathcal{C}_{\max})\log|\mathcal{C}|$.*

Theorem 2 *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 close to one [11, 20, 37].

4 Exact Algorithm

This section describes an exact pseudo-polynomial time dynamic programming algorithm for the CECD over taxonomy $\mathcal{X} = (R, \mathcal{C}, \mathcal{R})$. We assume that $u(C)$, $d(C)$, and $w(C)$ are positive integers for each $C \in \mathcal{C}$. In Section 8, we show that the algorithm can handle real values with scaling techniques at the expense of reporting near optimal solution instead of an optimal one. We refer to the second part of the queriability in Definition 2 as *free partition*. Given a concept $C \in \mathcal{X}$, the subtree rooted at C in $\mathcal{X}$ is denoted as $\mathcal{X}_C$, and the set of the children of C in $\mathcal{X}$ is denoted as $\texttt{child}(C)$. Let $Q[C, B, N]$ denote the maximum queriability over all designs in $\mathcal{X}_C$ such that the amount of queriability that these designs obtain from their free parts are exactly N, and they are selected by spending at most B units of the budget. Our algorithm computes $Q[C, B, N]$ for all $C \in \mathcal{X}$, all integers $0 \leq B \leq B_{\texttt{total}}$ and all integers $0 \leq N \leq N_{\texttt{total}}$, where $N_{\texttt{total}} = \sum_{C\in\texttt{leaf}(\mathcal{C})} u(C)d(C)$ and $\texttt{leaf}(\mathcal{C})$ is the set leaf nodes in $\mathcal{C}$. Our algorithm returns $\max_{0\leq N\leq N_{\texttt{total}}}(Q[R, B_{\texttt{total}}, N] + N)$ where R is the root of $\mathcal{X}$ and $B_{\texttt{total}}$ is the total budget. We try all possible queriabilities that one can obtain from the free part in $\mathcal{X}$ and N, to find out the maximum queriability over all designs in $\mathcal{X}$.

For a non-leaf concept C, we obtain a recursive description for $Q[C, B, N]$ according to the value of Q for the children of C. We consider the following two cases for C:

C is in the optimal design: If C belongs to the optimal design in $\mathcal{X}_C$, we have $N = 0$. This is because if C is picked, every concept in $\mathcal{X}_C$ belongs to a non-free partition in the selected design. In this case, we spend $w(C)$ of the budget to annotate C and the leftover budget $BL = B - w(C)$ should be assigned to the $\texttt{child}(C)$. Our algorithm tries all possible ways of assigning the leftover budget among the children of C. If C is selected, the total queriability obtained at C can be divided into the total non-free queriability of its children and the queriability of the partition of C. Consider the leaf nodes in $\mathcal{X}_C$ that belong to the free partitions in the subtrees rooted at the children of C. If C is selected, these nodes will be in the partition of C. Thus, the queriability of the partition of C is the total free queriability of the children of C times $\texttt{pr}(C)/d(C)$. Our algorithm tries all possible total free queriabilities for the children of C as well as all possible assignment of this free queriability to the set $\texttt{child}(C)$. Thus, we obtain the following recursion. We use Q_I to indicate the inclusion of C in the solution.

$$Q_I[C, B, N] = \max_{\mathcal{B},\mathcal{N}}\Big(\sum_{\texttt{Ch}\in\texttt{child}(C)} Q[\texttt{Ch}, \mathcal{B}(\texttt{Ch}), \mathcal{N}(\texttt{Ch})] + \frac{\texttt{pr}(C)}{d(C)}\sum_{\texttt{Ch}\in\texttt{child}(C)} \mathcal{N}(\texttt{Ch})\Big).$$

$\mathcal{B}$ includes all possible assignments of BL units of the budget to the children of C, and $\mathcal{N}$ includes all possible assignments of at most $N_{\texttt{total}}$ queriability to the children of C. If $N \neq 0$, we set $Q_I[C, B, N] = -\infty$ to indicate the infeasibility of a the case.

C is not in the optimal design. In this case, we want to obtain exactly N units of free queriability, while we maximize the total queriability. Our algorithm

tries all possible ways of assigning this N units into the children of C. Since C is not picked, the entire budget B can be spent on the subtrees rooted at the children of C. Hence, our algorithm tries all possible assignments of this B units of the budget among the children of C. Thus, we obtain the following recursions. We use Q_E to indicate the exclusion of C from the solution.

$$Q_E[C,B,N] = \max_{\mathcal{B},\mathcal{N}} \sum_{Ch \in \texttt{child}(C)} Q[Ch, \mathcal{B}(Ch), \mathcal{N}(Ch)]$$

$\mathcal{B}$ includes all possible assignments of B units of budget among the children of C, and $\mathcal{N}$ includes all possible assignments of exactly N unit of benefit from the queriability of free partitions among the children of C.

Our algorithm considers both cases above and returns the maximum queriability that can be obtained by either including or excluding C from the optimal solution:

$$Q[C,B,N] = \max(Q_I[C,B,N], Q_E[C,B,N]).$$

The leaf concepts are the base cases for our recursion. Let C be a leaf concept, and suppose that we want to optimize $Q[C,B,N]$. If $N = 0$, we must select C. Otherwise, we cannot select C. Thus, there are two cases to consider:
$\mathbf{N \neq 0}$: If $N = u(C)d(C)$, we set $Q[C,B,N] = 0$. Otherwise, we set $Q[C,B,N]$ to $-\infty$.
$\mathbf{N = 0}$: If $B \geq w(C)$, we set $Q[C,B,N] = \texttt{pr}(C)u(C)$. Otherwise, we set $Q[C,B,N]$ to $-\infty$.

This completes the explanation of our recursive algorithm. We turn this recursive algorithm into a dynamic programming to avoid redundant computations. To this end, we build a $|\mathcal{C}| \times B_{\texttt{total}} \times N_{\texttt{total}}$ table, and fill it according to our recursions. The running time of such a dynamic programming would be $O(|\mathcal{C}| B_{\texttt{total}} N_{\texttt{total}}) \cdot T_{\texttt{cell}}$, where $T_{\texttt{cell}}$ is the maximum time required to compute the value of a single cell disregarding recursive calls. $T_{\texttt{cell}}$ exponentially depends on the maximum degree of concepts in the taxonomy. Hence, as detailed below, we further optimize the running time of the algorithm by adding a preprocessing step to modify our input taxonomy to a binary tree (i.e. the maximum number of children of each concept is two). Assuming that our input taxonomy is a binary tree, for each $Q[C,B,N]$, we have $O(B_{\texttt{total}})$ ways of dividing B and $O(N_{\texttt{total}})$ ways of dividing the expected free queriability between the children of C. So, $T_{\texttt{cell}} = O(B_{\texttt{total}} N_{\texttt{total}})$. Let $D = \sum_{C \in \texttt{leaf}(\mathcal{C})} d(C)$ and $U = \sum_{C \in \texttt{leaf}(\mathcal{C})} u(C)$. By Cauchy-Schwartz inequality, we have $N_{\texttt{total}} \leq UD$. We conclude that the running time is $O(|\mathcal{C}|(B_{\texttt{total}} N_{\texttt{total}})^2) = O(|\mathcal{C}|(B_{\texttt{total}} UD)^2)$.

Transforming a taxonomy into binary taxonomies. We explain how to transform an arbitrary taxonomy to a binary taxonomy. Let C be a non-leaf concept in $\mathcal{X}$. We replace the induced subtree of $C \cup \texttt{child}(C)$ with a full binary tree $\mathcal{X}'_C$ whose root is C and whose leaves are $\texttt{child}(C)$ as shown in Figure 8. Some internal nodes of $\mathcal{X}'_C$ do not correspond to any node in $\mathcal{X}$. We refer to such internal nodes as *dummy nodes*, and set their cost to $B_{\texttt{total}} + 1$ to make sure that our algorithm does not include them in the output design.

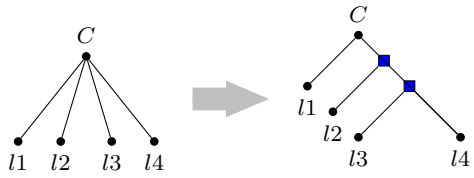

Fig. 8 Transforming an input taxonomy. Dummy nodes are in blue squares.

Applying the aforementioned transformation to all nodes of $\mathcal{X}$, we obtain a binary taxonomy $\mathcal{X}' = (R, \mathcal{C}', \mathcal{R}')$. The number of nodes in $\mathcal{C}'$ is at most twice the number of nodes in $\mathcal{C}$. Therefore, we can find the optimal design for $\mathcal{X}'$ in the same asymptotic running time, $O(|\mathcal{C}|(B_{\texttt{total}} UD)^2)$. Since this transformation does not change the subset of leaf concepts in the subtree rooted in any internal node, any internal node in $\mathcal{X}$ corresponds to a solution in $\mathcal{X}'$ with the same cost and queriability. Since dummy nodes are too costly, they do not introduce any new solution.

Theorem 3 *There is an exact algorithm to solve the* CECD *problem over taxonomy* $\mathcal{X} = (R, \mathcal{C}, \mathcal{R})$ *with budget* $B_{\texttt{total}}$ *in* $O(|\mathcal{C}| B^2_{\texttt{total}} U^2 D^2)$.

5 CECD with Cost Dependency

The cost of extracting a concept may vary if its ancestors in the taxonomy have been already annotated. For example, if the instances of *person* have been already annotated in a collection, the extractor of *artist* will examine only a subset of the collection that are annotated by concept *person*, which may result in a faster extraction. Also, the extractors for *artist* and *person* may share some annotation rules and/or features. We generalize the CECD problem so that the cost of annotating a concept in a taxonomy may change depending on whether its ancestors in the taxonomy are part of the design. One may think of more complex cost functions that represent dependencies between arbitrary concepts in the taxonomy. However, it requires a space and time exponential in terms of the number of concepts in the taxonomy to express and estimate such functions. One may reduce the amount of space and effort to store and

estimate the cost values by enforcing some restrictions. On the other hand, finding such restrictions requires extensive empirical studies and more space than a single paper and is a subject of future work.

Assume that multiple ancestors of a concept C in $\mathcal{X} = (R, \mathcal{C}, \mathcal{R})$ are in the selected design. The annotator of C may examine only the documents annotated by the closest ancestor of C in the design. Moreover, the closest ancestor of C may share more annotation rules /features with C than other ancestors of C. Hence, we consider only the dependency between cost of C and its closest selected ancestor in $\mathcal{X}$. More formally, for each concept C, let $\texttt{anc}(C)$ be a positive integer value indicating the number of edges between C and its closest ancestor in the design. We redefine the cost w as a real-valued function over pairs of concepts and positive integer values. The total cost of design S over $\mathcal{X}$ is $\sum_C w(C, \texttt{anc}(C))$. We define the problem of CECD *with cost dependency* exactly the same as Problem 1, by only replacing its cost function with the redefined cost function. Because CECD is a special instance of the problem of CECD with cost dependency, CECD with cost dependency is also **NP**-hard.

We extend the dynamic programming algorithm in Section 4 to design a pseudo-polynomial time algorithm for CECD with cost dependency. As before, we assume that the values of functions u, d and w are positive integers. We define $\mathcal{X}_C$ and $\texttt{leaf}()$ the same way that they are defined in Section 4. Let $Q[C, B, N, \texttt{anc}]$ denote the maximum queriability we can obtain constraint to budget B from the partitions in $\mathcal{X}_C$ such that the queriability of the free parts is N and $\texttt{anc}$ indicates the closest selected ancestor of C in the design. The algorithm computes values of Q for all budgets up to total budget $B_{\texttt{total}}$ and all free queriability N up to $N_{\texttt{total}} = \sum_{C \in \texttt{leaf}(\mathcal{C})} u(C)d(C)$ and returns

$$\max_{0 \le N \le N_{\texttt{total}}} (Q[R, B_{\texttt{total}}, N, 1] + N)$$

where R is the root of $\mathcal{X}$ and $B_{\texttt{total}}$ is the total budget. Note that for the root node, R, the exact value of $\texttt{anc}$ does not matter because R has no ancestor. We use the technique in Section 4 to transform the taxonomy to a binary tree. We extend the recursive formulas in Section 4 for $Q[C, B, N, \texttt{anc}]$ in the following two cases. Q_I and Q_E are defined similar to the ones in Section 4.

C is in the optimal design.

$$Q_I[C, B, N, \texttt{anc}] = \max_{\mathcal{B}, \mathcal{N}} \Big(\sum_{\texttt{Ch} \in \texttt{child}(C)} Q[\texttt{Ch}, \mathcal{B}(\texttt{Ch}), \mathcal{N}(\texttt{Ch}), 1] + \frac{\texttt{pr}(C)}{d(C)} \sum_{\texttt{Ch} \in \texttt{child}(C)} \mathcal{N}(\texttt{Ch}) \Big).$$

$\mathcal{B}$ includes all possible assignments of $B - w(C, \texttt{anc})$ units of the budget, and $\mathcal{N}$ includes all possible assignments of at most N free queriability between children of C. Because concept C is in the optimal design, it resets the value of *anc* to 1 for its children.

C is not in the optimal design. If C is **not** a dummy node:

$$Q_E[C, B, N, \texttt{anc}] = \max_{\mathcal{B}, \mathcal{N}} \sum_{Ch \in \texttt{child}(C)} Q[Ch, \mathcal{B}(Ch), \mathcal{N}(Ch), \texttt{anc} + 1]$$

Since C is not selected, it updates and transfers the information about its closest selected ancestor to its children. If C is a dummy node:

$$Q_E[C, B, N, \texttt{anc}] = \max_{\mathcal{B}, \mathcal{N}} \sum_{Ch \in \texttt{child}(C)} Q[Ch, \mathcal{B}(Ch), \mathcal{N}(Ch), \texttt{anc}]$$

The dummy nodes are not selected and do not change the value of $\texttt{anc}$. In these formulas, $\mathcal{B}$ includes all possible assignments of B units of the budget and $\mathcal{N}$ includes all possible assignments of exactly N free queriability between children of C.

The leaf concepts make the base cases for our recursion and their equations are similar to the bases cases in Section 4. Because the information about the closest selected ancestor of a leaf node is available in Q, the algorithm can compute the cost of leaf node.

Let L be the height of $\mathcal{X}$. To compute the running time of this algorithm, we need to give an upper bound on the number of cells in Q and the time required to compute the value for each cell. The time to compute a single cell in Q is exponential in terms of the maximum degree of the taxonomy, which is 2 after transforming $\mathcal{X}$ to a binary tree. Because we have $N \le UD$, and the maximum value of $\texttt{anc}$ is L, the time for computing all cells in Q will be $O(|\mathcal{C}|(B_{\texttt{total}}UD)^2L)$. Generally, L for most real-world taxonomy is considerably smaller than $|\mathcal{C}|$ and closer to $O(\log |\mathcal{C}|)$.

6 Queries With Multiple Concepts

In this section, we propose fast algorithms for the problem of CECD where queries may refer to multiple concepts. We call this problem, Multiple-Concept CECD (MC-CECD).

6.1 MC-CECD Over a Set of Concepts

We assume that each query refers to at most two different concepts. The choice of two is only for the sake of

simplicity and our algorithm extends to any fixed number of concepts. We also assume that the annotation each (leaf) concept over a dataset is independent of the annotation of any other (leaf) concept in the dataset to simplify our analysis. Our empirical studies in Section 8 indicate that generally this assumption does *not* significantly reduce the accuracy of our models and algorithms. Let $QU_n(\mathcal{S})$ be the queriability of $\mathcal{S}$ from a set of concepts $\mathcal{C}$ over queries with exactly n concepts. The queriability of $\mathcal{S}$ for queries with one concept, $QU_1(\mathcal{S})$, is $\sum_{C\in\mathcal{S}} u(C)\mathtt{pr}(C)+\sum_{C\in\mathtt{free}(\mathcal{S})} u(C)d(C)$ [50]. Next, we compute $QU_2(\mathcal{S})$. Assume that query q refers to entities of concepts C_1 and C_2. The query interface should select the answers to q from the set of documents that contain instances of both concepts. If the instances of both concepts are annotated in the dataset, the query interface will correctly identify such documents. If only C_1 is annotated, the query interface will return documents annotated as C_1, of which only about $d(C_2)$ may contain the desired answers. If none of these concepts are annotated, the query interface has to explore all documents in the collection of which only about $d(C_1 \cap C_2)$ may have the desired answers. We have:

$$\begin{aligned} QU_2(\mathcal{S}) = &\sum_{\substack{C_1\in\mathcal{S}\\ C_2\in\mathcal{S}}} u(C_1\cap C_2)\mathtt{pr}(C_1)\mathtt{pr}(C_2)\\ &+\sum_{\substack{C_1\in\mathcal{S}\\ C_2\in\mathtt{free}(\mathcal{S})}} u(C_1\cap C_2)d(C_2)\mathtt{pr}(C_1)\\ &+\sum_{\substack{C_1\in\mathtt{free}(\mathcal{S})\\ C_2\in\mathtt{free}(\mathcal{S})}} u(C_1\cap C_2)d(C_1\cap C_2) \end{aligned}$$

The total queriability of $\mathcal{S}$, $QU(\mathcal{S})$, is $QU_1(\mathcal{S})+QU_2(\mathcal{S})$. In MC-CECD, given a set of concepts $\mathcal{C}$, dataset DS in the domain of $\mathcal{C}$ and budget B, we like to find design $\mathcal{S}$ over $\mathcal{C}$ such that $\sum_{C\in\mathcal{S}} w(C) \le B$ and $\mathcal{S}$ has the maximum queriability. We reduce MC-CECD to CECD over a set of concepts by considering instances in which $u(C_i \cap C_j) = 0$ for all C_i, C_j. Hence, MC-CECD over a set of concepts is **NP**-hard.

Our algorithm for MC-CECD is based on the algorithm of Set Union Knapsack (SU-Knapsack) problem [5]. In SU-Knapsack, we have a collection of elements $\mathcal{E} = \{e_1, ..., e_n\}$, each with a *positive weight* denoted by $w(e_i)$. Furthermore, we have a collection of *items* $\mathcal{I} = \{i_1, ..., i_m\}$ such that each item is a subset of $\mathcal{E}$. Each item i has *profit* $p(i)$. Given budget B, the goal of SU-Knapsack$(\mathcal{E},\mathcal{I})$ is to find a set of items $\mathcal{I}_{\text{sol}}$ such that the total cost of the elements in $\mathcal{I}_{\text{sol}}$, $\sum_{e\in\bigcup_{i\in\mathcal{I}_{\text{sol}}}} w(e)$, is not more than B, and the total profit of $\mathcal{I}_{\text{sol}}$, $\sum_{i\in\mathcal{I}_{\text{sol}}} p(i)$, is maximized. The algorithm proposed in [5] starts with all possible pair of items as its initial set. Then, for each set, it goes through several iterations and at each iteration picks an item i from the remaining items with the maximum value of $\frac{p(i)}{w'(i)}$, where $w'(i) = \sum_{e\in i} \frac{w(e)}{\mathtt{freq}(e)}$ and $\mathtt{freq}(e)$ denotes the number of occurrences of element e in items of $\mathcal{I}$. The iterations will be performed until the budget is used up. The algorithm keeps the solution with maximum benefit amongst all constructed solutions. Then, it compares this solution with the solution that has only the item with maximum profit and returns the one with the maximum profit amongst them.

Consider an instance of MC-CECD over concepts $\{C_1, ..., C_n\}$ with budget B. To solve this instance, we make an instance $\mathcal{M}$ of SU-Knapsack as follows. Let $C_1, ..., C_n$ be the input set of elements such that $w(C_i)$ is the annotation cost of concept C_i. We define the input items to be all subsets of $\{C_1, ..., C_n\}$ of size at most two (including empty set). The profit of each item is:

$$\begin{aligned} p(\emptyset) &= \sum_{C\in\mathcal{C}} u(C)d(C)\\ &\quad+\sum_{C_1,C_2\in\mathcal{C}} u(C_1\cap C_2)d(C_1\cap C_2)\\ p(\{C_i\}) &= u(C_i)(\mathtt{pr}(C_i)-d(C_i))\\ &\quad+\sum_{C\in\mathcal{C}} u(C_i\cap C)(d(C)\mathtt{pr}(C_i)-d(C\cap C_i))\\ p(\{C_i,C_j\}) &= u(C_i\cap C_j)\mathtt{pr}(C_i)\mathtt{pr}(C_j)\\ &\quad+u(C_i\cap C_j)d(C_i\cap C_j)\\ &\quad-u(C_i\cap C_j)d(C_i)\mathtt{pr}(C_j)\\ &\quad-u(C_i\cap C_j)\mathtt{pr}(C_i)d(C_j) \end{aligned}$$

The item corresponding to the empty set has cost zero and is picked by every solution for $\mathcal{M}$. Because annotators work better than a random selection, for each concept C, $\mathtt{pr}(C) \ge d(C)$ [37]. Hence, the profits of items in $\mathcal{M}$ are positive.

Theorem 4 *There exists a* $(1-1/e^{\frac{1}{k+1}})$*-approximation algorithm for* MC-CECD *instances in which each concept queried with at most* k *different other concepts.*

6.2 MC-CECD Over Tree Taxonomies

Given design $\mathcal{S}$ over tree taxonomy $\mathcal{X} = (\mathcal{C}, \mathcal{R}, R)$, we denote the queriability of $\mathcal{S}$ for queries with exactly one concept and exactly two concepts by $QU_1(\mathcal{S})$ and $QU_2(\mathcal{S})$, respectively. Definition 2 computes $QU_1(\mathcal{S})$ over $\mathcal{X}$. Next, we compute the value of $QU_2(\mathcal{S})$. Let query q refers to entities of concepts C_1 and C_2. Similar to computing $QU_2(\mathcal{S})$ over set of concepts, there are four cases to consider. First, if C_1 and C_2 both belong to the same partition P in $\mathcal{S}$, the query interface may examine only the documents annotated

with P to find answers for q. The documents that contain instances of both C_1 and C_2 contain relevant answers to q. Hence, the query interface will find the desired answers to q with the probability of $\frac{d(C_1 \cap C_2)}{d(P)}$. In the second case, C_1 and C_2 belong to different partitions P_1 and P_2 in $\mathcal{S}$, respectively. The query interface should search the documents annotated under both P_1 and P_2, which are $d(P_1 \cap P_2)$ portion of the collection. Because the desired answers should contain instances of both concepts, the query interface finds the desired answers to q with probability of $\frac{d(C_1 \cap C_2)}{d(P_1 \cap P_2)}$. Now, assume that only one of C_1 or C_2, e.g., C_1, is in a partition P of $\mathcal{S}$. The query interface may search the documents that contain annotations of P. Because the desired answers are documents with instances of both C_1 and C_2, the query interface may return the desired answer with the probability of $\frac{d(C_1 \cap C_2)}{d(P)}$. Finally, if neither C_1 nor C_2 are in any partition, i.e., they both belong to $\texttt{free}(\mathcal{S})$, the query interface will search the whole collection and may find the desired answer in $d(C_1 \cap C_2)$ portion of the collection. We have:

$$
\begin{aligned}
&QU_2(\mathcal{S}) = \\
&\sum_{P \in \mathcal{S}} \sum_{\substack{C_1 \in \texttt{part}(P) \\ C_2 \in \texttt{part}(P)}} \frac{u(C_1 \cap C_2)\ d(C_1 \cap C_2)}{d(P)} \texttt{pr}(P) \\
&+ \sum_{\substack{P_1 \in \mathcal{S} \\ P_2 \in \mathcal{S}}} \sum_{\substack{C_1 \in \texttt{part}(P_1) \\ C_2 \in \texttt{part}(P_2)}} \frac{u(C_1 \cap C_2) d(C_1 \cap C_2)}{d(P_1 \cap P_2)} \texttt{pr}(P_1)\texttt{pr}(P_2) \\
&+ \sum_{P \in \mathcal{S}} \sum_{\substack{C_1 \in \texttt{part}(P) \\ C_2 \in \texttt{free}(\mathcal{S})}} \frac{u(C_1 \cap C_2) d(C_1 \cap C_2)}{d(P)} \texttt{pr}(P) \\
&+ \sum_{\substack{C_1 \in \texttt{free}(\mathcal{S}) \\ C_2 \in \texttt{free}(\mathcal{S})}} u(C_1 \cap C_2) d(C_1 \cap C_2)
\end{aligned}
$$

Let $QU(\mathcal{S})$ denote the queriability of $\mathcal{S}$ over queries with at most two concepts, i.e., $QU(\mathcal{S}) = QU_1(\mathcal{S}) + QU_2(\mathcal{S})$. We define the problem of MC-CECD over tree taxonomies similar to Problem 1 with the new formula for $QU(\mathcal{S})$. Since CECD over tree taxonomies is **NP**-hard, MC-CECD is **NP**-hard (consider instances with $u(C_i \cap C_j) = 0$ for all C_i, C_j).

We extend the LEVEL-WISE algorithm for the problem of MC-CECD over *tree* taxonomy. To this end, we combine the LEVEL-WISE algorithm and the algorithm for SU-KNAPSACK described in Section 6.1. Consider a level r of the tree taxonomy and let $C_1^r, \cdots, C_p^r$ be the concepts of level r. We compute the profits as defined in Section 6.1 for all subsets of size at most two (including the empty set) of the concepts in level r. Then, we find the solution with maximum queriability in level r using the approximation algorithm of SU-KNAPSACK. Our algorithm returns the solution with the maximum queriability over all levels of the taxonomy. The running time of the presented heuristic is $O(|\mathcal{C}|^3 L) \sim O(|\mathcal{C}|^3 \log |\mathcal{C}|)$ where L is the height of the tree taxonomy.

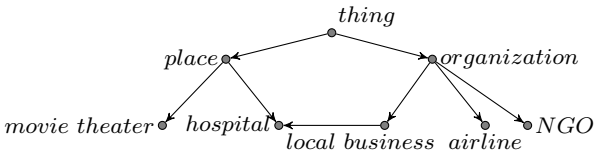

Fig. 9 Fragments of *schema.org* taxonomy.

7 Cost-Effective Design for DAG Taxonomies

7.1 Directed Acyclic Graph Taxonomies

Many taxonomies are in form of *directed acyclic graphs* (DAGs). Figure 9 shows fragments of *schema.org* taxonomy. Some concepts in this taxonomy are included in multiple superclasses. For example, a *hospital* is both a *place* and an *organization*. Therefore, a tree structure is not able to represent these relationships.

Formally, a *directed acyclic graph taxonomy* $\mathcal{X} = (R, \mathcal{C}, \mathcal{R})$ (*DAG taxonomy* for short), is a DAG, with vertex set $\mathcal{C}$, edge set $\mathcal{R}$ and root R where $\mathcal{C}$ is a set of concepts, $(D, C) \in \mathcal{R}$ if and only if $D, C \in \mathcal{C}$ and C is a subclass of D. The root concept $R \in \mathcal{X}$ is a concept without superclass. A concept $C \in \mathcal{C}$ is a *leaf concept* if and only if it has no subclass in $\mathcal{X}$. The definitions of *child*, *ancestor* and *descendant* over tree taxonomies naturally extends to DAGs.

7.2 Design Queriability

Design $\mathcal{S}$ over DAG taxonomy $\mathcal{X} = (R, \mathcal{C}, \mathcal{R})$ is a non-empty subset of $\mathcal{C} \setminus \{R\}$. Because of the richer structure of DAG taxonomies, designs over DAG taxonomies may improve the effectiveness of answering queries in more ways than the ones over tree taxonomies. For example, let dataset DS be in the domain of DAG taxonomy in Figure 9, and $\mathcal{S}_1 = \{$*place*, *organization*$\}$ be a design over this taxonomy. Because concept *hospital* is a subclass of both *place* and *organization*, its entities in DS are annotated by both these concepts. By examining the entities that are annotated by both *place* and *organization*, the query interface is able to identify the instances of *hospital* in DS. Generally, the query interface may pinpoint instances of some concepts in the dataset by considering the intersections of multiple con-

cepts in a design over a DAG taxonomy. Hence, subsets of a design may create partitions in a DAG taxonomy.

Definition 3 Let $\mathcal{S}$ be a design over the DAG taxonomy $\mathcal{X} = (R, \mathcal{C}, \mathcal{R})$, and let $C \in \mathcal{C}$ be a leaf concept. An ancestor A of C in $\mathcal{S}$ is C's direct ancestor if and only if one of the followings hold.
- $A = C$.
- For each $D \in \mathcal{S}$, if D is an ancestor of C then D is not a descendant of A.

The *full-ancestor-set* of C is the set of *all its direct ancestors*.

Example 5 Consider a DAG taxonomy shown in Figure 9. The set {*place*, *organization*} is the full-ancestor-set of the concept *hospital* in design $\mathcal{S}_1$ = {*place*, *organization*}, and the set {*local business*, *place*} is the full-ancestor-set of the concept *hospital* in design $\mathcal{S}_2$ = {*organization*, *local business*, *place*} over the taxonomy in Figure 9.

Definition 4 Given design $\mathcal{S}$ over the DAG taxonomy $\mathcal{X} = (R, \mathcal{C}, \mathcal{R})$, the partition of a set of concepts $\mathcal{D} \subseteq \mathcal{S}$ is a set of leaf concepts $\mathcal{L} \subseteq \mathcal{C}$ such that for every leaf concept $L \in \mathcal{L}$, $\mathcal{D}$ is the full-ancestor-set of L.

Example 6 Let $\mathcal{S}_1$ = {*organization*, *place*} be a design over the DAG taxonomy shown in Figure 9. The concept *hospital* belongs to the partition of {*organization*, *place*} in $\mathcal{S}_1$. However, it does not belong to the partition of {*place*} because {*place*} is not the full-ancestor-set of *hospital*.

The definitions of functions `part` and `free` over tree taxonomies naturally extend to DAG taxonomies. We define the frequency of partition P, denoted by $d(P)$, as the frequency of the intersection of concepts in its root. Using a similar analysis to the one in Section 2.3, we define the queriability of design S over DAG taxonomy $\mathcal{X} = (R, \mathcal{C}, \mathcal{R})$ as $\sum_{P \in \texttt{all-parts}(\mathcal{S})} \frac{\sum_{C \in P} u(C)d(C)}{d(P)} + \sum_{C \in \texttt{free}(\mathcal{S})} u(C)d(C)$ where the function $\texttt{all-parts}(\mathcal{S})$ returns the collection of all full-ancestor-sets of $\mathcal{S}$ in $\mathcal{X}$.

7.3 Hardness of CECD over DAG Taxonomies

We define the CECD problem over DAG taxonomies similar to the CECD over tree taxonomies. Following from the **NP**-hardness results for CECD problem over tree taxonomy, CECD problem over DAG taxonomies is NP-hard. We prove that finding an approximation algorithm with a reasonably small bound on its approximation ratio for the problem CECD over DAG taxonomies is significantly hard. Unfortunately, this is true even for the special cases where concepts in the taxonomy have equal costs or the desired design is disjoint. More precisely, we show that the CECD problem over a DAG taxonomy generalizes a hard problem in the approximation algorithms literature: Densest-k-Subgraph. Given a graph $G = (V, E)$, in the Densest-k-Subgraph problem, the goal is to compute a subset $U \in V$ of size k that maximizes the number of edges in the induced subgraph of U. It is known that, unless **P** = **NP**, no polynomial time approximation scheme, i.e., PTAS, exists to compute the densest subgraph [32]. Also, there are strong evidences that Densest-k-Subgraph does not admit any approximation guarantee better than polylogarithmic factor [4, 6]. Recently, it has been shown that, assuming exponential time hypothesis (ETH), there is no polynomial time algorithm that approximates Densest-k-Subgraph within a factor of $n^{1/(\log\log n)^c}$ [36]. The following theorem shows that approximating the k-densest subgraph reduces to approximating CECD.

Theorem 5 *An α-approximation algorithm for the* CECD *problem over DAG taxonomy with m concepts implies that there is an algorithm for the* Densest-k-Subgraph *problem on $G = (V, E)$ with n vertices that returns a $O(\alpha)$-approximate solution.*

Because the concepts in the instance of CECD in the proof of Theorem 5 have equal costs and its optimal solution is disjoint, i.e., there is no directed path between any two of concepts in the design, the hardness results of Theorem 5 is true for the special cases of CECD over DAG taxonomies where the concepts are equally costly and the problem has disjoint optimal solution.

The following corollaries result from Theorem 5.

Corollary 1 *If* **NP** $\not\subseteq \cap_{\epsilon>0}$**BPTIME**$(2^{n^\epsilon})$, *there is no polynomial time approximation scheme,* **PTAS**, *for* CECD *over DAG taxonomies.*

Corollary 2 *Assuming Exponential Time Hypothesis (ETH), the problem of* CECD *over DAG taxonomies does not admit an approximation guarantee better than* $n^{1/(\log\log n)^c}$.

8 Experiments

8.1 Experiment Setting

8.1.1 Taxonomies and datasets

Taxonomies: We have extracted eight taxonomies from YAGO ontology version 2008-w40-2 [49] to validate our

Table 2 The sizes and heights of taxonomies and the sizes of corresponding datasets and query workloads.

Taxonomy	T1	T2	T3	T4	T5	T6	T7	T8
#Concept	10	17	17	28	63	185	279	2269
#Height	2	2	3	7	6	8	8	9
#Documents	68982	267653	88479	1470661	1470661	1470661	1470661	1470661
#queries	648	256	146	4219	4888	4728	5216	11156

model and evaluate effectiveness and efficiency of our proposed algorithms. YAGO organizes its concepts using subclass relationships in a DAG with a single root. We have created the breath-first tree of YAGO and randomly selected the concepts from the tree for our taxonomies. To validate our model, we have to compute and compare the effectiveness of answering queries using every feasible design over a taxonomy. Thus, we need tree taxonomies with relatively small number of concepts for our validation experiments. We have extracted three tree taxonomies with relatively small numbers of nodes, called *T1*, *T2* and *T3*, to use in our validation experiments. The selected concepts for T1, T2 and T3 are from level 3 to 6 of the full YAGO tree which has a total of 9 levels. We have further picked five other tree taxonomies with larger numbers of concepts, denoted as *T4*, *T5*, *T6*, *T7* and *T8*. The concepts for T4 and T5 are selected from level 3 to 6 of the full YAGO tree. The concepts for T6 and T7 are randomly selected from levels 2 to 9 of the full YAGO tree. T8 contains all concepts from the original YAGO taxonomy that appear at least once in our dataset, which is the collection of English Wikipedia articles. We use all tree taxonomies to evaluate the effectiveness and efficiency of our proposed algorithms. Table 2 shows the properties of these taxonomies.

Datasets: We have used the collection of English Wikipedia articles from the Wikipedia dump (*dumps.wikimedia.org*) of October 8, 2008 that is annotated by concepts from YAGO ontology in our experiments [49]. This collection originally contains 2,666,190 articles of which 1,470,661 articles are annotated by at least a concept from YAGO ontology. For each taxonomy in our set 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 2 shows the properties of these eight datasets. The annotation accuracies of the concepts in selected taxonomies over these datasets are between 0.8 and 0.95.

8.1.2 Query Workload

We use a subset of Bing (*bing.com*) query log whose relevant answers are Wikipedia articles [14]. The relevant answers for each query in this query workload have been determined using the click-through information by eliminating noisy clicks. Each query contains between one to six keywords and has between one to two relevant answers with most queries having one relevant answer. Because the query log does not have the concepts behind its queries, we adapt an automatic approach to find the leaf concept from the taxonomy associated with each query. We label each query by the concept of the matching instance in its relevant answer(s). Then, we select queries labeled with a *single* concept. Using this method, we create a query workload per each of our datasets. It is well known that the effectiveness of answering some queries may not be improved by annotating the dataset [44]. For instance, all candidate answers for a query may contain mentions to the entities of the query concept. To reasonably evaluate our algorithms, we have ignored the queries whose rankings remain the same over the unannotated version and the version of the dataset where all concepts in the taxonomy are annotated. Table 2 shows the information about our query workloads.

We estimate the popularities of concepts for each taxonomy by sampling a small subset of randomly selected queries from their corresponding query workloads. We compute the popularity of each concept using estimation error rate of 5% under the 95% confidence level. The number of sampled queries are between 100 and 500 for each taxonomy. Because some concepts in a taxonomy may not appear in its query workload, we smooth the popularity of a concept C, $u(C)$, using the Bayesian m-estimate method [35]: $\hat{u}(C) = \frac{\hat{P}(C|QW)+mp}{m+\sum_C \hat{P}(C|QW)}$, where $\hat{P}(C|QW)$ is the probability that C occurs in the query workload QW and p denotes the prior probability. We set the value of the smoothing parameter, m, to 1 and use a uniform distribution for all the prior probabilities.

8.1.3 Query Interface

We index our datasets using Lucene (*lucene.apache.org*) Given a query, we rank its candidate answers using BM25 ranking formula, which is shown to be more effective than other similar document ranking methods [35]. 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 in the collection, the query interface returns the list of documents 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 100 GB of main memory and two quad core processors.

8.1.4 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 mean reciprocal rank (MRR) [35]. MRR is an inverse of the rank of the first relevant answer in the returned list of answers. MRR is used to measure the effectiveness of results for queries that have a small number of relevant answers [35]. Since most queries in our query workload have a single relevant answer, MRR is an appropriate metric to measure the effectiveness of their results. We measure the statistical significance of our results using the paired-t-test at a significant level of 0.05. We also report the precision at top 3 answers for our results in the longer version of this paper [51]. Generally, the results of MRR and precision at top 3 follow similar trends.

8.1.5 Cost Models

We use three 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. Second, when 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. These two cost models have also been used to model the costs of large-scale data curation [19, 43, 50]. Lastly, the cost annotating instances of a concept may depend on how frequent the instances of the concept appear in the dataset. Hence, in our third cost model called *frequency-based cost* model, we set that a cost of annotating a concept c to $d(c)/\sum_{l \text{ is leaf}} d(l)$. 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 leaf concepts in a taxonomy and 0 means no budget is available.

8.2 Validating Queriability Function

Oracle: Given a fixed budget, Oracle enumerates all feasible designs over the input taxonomy. Given an effectiveness metric, such as MRR, for each design, it computes the average the effectiveness metric for all queries in the query workload over the dataset annotated by the design. It then returns the design with maximum value of the average effectiveness metric.

Popularity Maximization (*PM*): Following the traditional approach towards conceptual design for databases, one may select concepts in a design that are *more important* for users [23]. Hence, we implement an algorithm, called *PM*, that enumerates all feasible designs, such as $\mathcal{S}$, in a taxonomy and returns the one with the maximum value of $\sum_{p \in \texttt{part}(\mathcal{S})} \sum_{C \in p} u(C)\texttt{pr}(p)$. This design contains the concepts that are more frequently queried by users and also annotated more accurately.

Queriability Maximization (*QM*): QM enumerates all feasible designs over the input taxonomy and returns the one with the maximum queriability as computed in Section 2.3.

Because we would like to explore how accurately PM and QM predict the amount of improvement in the effectiveness of answering queries by a design, we assume that these algorithms have complete information about the popularities and frequencies of concepts. Since Oracle, PM and QM algorithms enumerate all feasible designs, it is not possible to run them over large taxonomies. Hence, we run these algorithms over small taxonomies, i.e. T1, T2 and T3.

Table 3 illustrates the average MRR achieved by Oracle, QM and PM over taxonomies T1, T2 and T3 for various budgets. We do not report the values of average MRR for budgets greater than 0.7 for T1 and T2 and 0.6 for T3 because all algorithms are equally effective. The values of MRR show at budget 0.0 is the one achieved by BM25 ranking without annotating any concept in the datasets. We note that there is no improvement in average MRR over taxonomy T1 for budget 0.1 under uniform cost because each concept in the taxonomy costs more than 0.1.

Over all taxonomies and cost models, the designs selected by QM deliver close MRR values to the ones selected by Oracle. There are a few cases where the results of QM are significantly worse than the results of Oracle. For instance, consider the results of QM and Oracle for budget 0.3 over taxonomy T2. In T2, concept *writing* is the parent of a leaf concept *dramatic composition* and a couple other leaf concepts whose popularities are much less than that of *dramatic composition*. QM picks a design that contains *dramatic composition*. This design will deliver the highest values of MRR for queries with

Table 3 Average MRR for Oracle, PM and QM over T1, T2 and T3. Statistically significant differences between PM and QM and between Oracle and QM are marked in bold and italic, respectively. B denotes a given budget.

		Uniform Cost			Random Cost			Frequency-based Cost		
	B	Oracle	QM	PM	Oracle	QM	PM	Oracle	QM	PM
T1	0.0	0.187								
	0.1	0.187	0.187	0.187	0.259	0.255	0.239	0.475	0.475	0.475
	0.2	0.362	**0.362**	0.197	0.385	**0.384**	0.226	0.475	0.475	0.475
	0.3	0.415	**0.406**	0.203	0.424	**0.417**	0.212	0.475	0.475	0.475
	0.4	0.459	**0.459**	0.227	0.461	**0.461**	0.262	0.475	0.475	0.475
	0.5	0.492	**0.492**	0.400	0.492	**0.492**	0.341	0.475	0.475	0.475
	0.6	0.501	**0.501**	0.444	0.501	**0.499**	0.426	0.475	0.475	0.475
	0.7	0.507	0.507	0.497	0.507	**0.504**	0.476	0.475	0.475	0.475
T2	0.0	0.342								
	0.1	0.504	0.479	0.504	0.515	0.471	0.482	0.598	0.598	0.598
	0.2	0.577	0.543	0.551	0.616	0.586	0.559	0.662	0.662	0.662
	0.3	*0.641*	**0.629**	0.574	0.677	**0.661**	0.576	0.677	0.674	0.674
	0.4	0.729	**0.729**	0.586	0.725	**0.725**	0.616	0.741	0.741	0.741
	0.5	0.745	**0.745**	0.615	0.744	**0.742**	0.652	0.747	0.747	0.747
	0.6	0.751	**0.751**	0.647	0.754	0.754	0.695	0.748	0.748	0.748
	0.7	0.763	0.763	0.759	0.763	0.758	0.732	0.748	0.748	0.748
T3	0.0	0.322								
	0.1	0.469	0.469	0.453	0.528	0.521	0.514	0.447	0.447	0.447
	0.2	0.595	0.594	0.579	0.646	**0.629**	0.587	0.531	0.531	0.531
	0.3	0.680	**0.679**	0.600	0.714	**0.712**	0.660	0.594	0.594	0.594
	0.4	0.745	**0.734**	0.685	0.747	0.739	0.711	0.725	0.725	0.725
	0.5	0.754	0.754	0.741	0.758	0.757	0.748	0.734	0.734	0.734
	0.6	0.760	0.760	0.760	0.760	0.760	0.760	0.734	0.734	0.734

concept *dramatic composition*, but it does *not* help improving the values of MRR for queries whose concepts are other children of *writing* over the unannotated dataset. That is, QM picks a relatively less popular concept, but maximizes the improvement of the effectiveness for queries with this concept. On the other hand, Oracle selects *writing* instead of *dramatic composition*. Intuitively, this design improves the values of MRR for queries with concept *dramatic composition* less than the design selected by QM. However, this design will improve the values of MRR for queries with other child concepts of *writing*. For this dataset, selecting *writing* helps improving the values of MRR for queries with concept *dramatic composition* as equal as selecting *dramatic composition*. Because, the design selected by QM is not able to improve the effectiveness of answering queries whose concepts are other children of *writing*, QM is less effective than Oracle. We have observed a similar behavior for other cases when the results of QM are significantly worse than Oracle. This observation suggests that if the budget is relatively small, it is sometimes better to annotate rather general concepts.

Oracle, QM and PM are equally effective using frequency-based cost model. Under this cost model, each leaf concept is less costly than its ancestors. As we have explained in Section 1, in this situation, every algorithm chooses leaf concepts. Thus, all algorithms return the same design for each budget. Furthermore, Oracle, QM and PM return the same values of average MRR over T1 for all budgets. The distribution of frequencies in the leaf concepts of T1 is very skewed in which concept *person* has frequency of 0.92 and the other 8 leaf concept have total frequency of 0.1. Using budgets 0.1-0.9, all algorithms pick 8 out of 9 leaf concepts in T1. Because the ancestors of *person* are more costly than *person*, none of the algorithms can pick *person*'s ancestors using budgets 0.1-0.9. Therefore, over T1, the algorithms are equally effective across all test budgets under frequency-based cost model.

Nevertheless, the results from Table 3 indicate that QM delivers designs that improve the average MRR of answering queries more than those delivered by PM. Overall, PM annotates more general concepts from the taxonomy to improve the effectiveness of larger number of queries. Hence, to answer a query, the query interface often has to examine the documents annotated by an ancestor of the query concept. As this set of documents contain many answers whose concepts are different from the query concept, the query interface is usually not able to improve the value of MRR for a query significantly. QM selects the designs with relatively less general concepts. Although its designs may not improve the ranking quality of every query, the designs significantly improve the ranking quality of queries whose concepts belong to the selected designs.

8.3 Effectiveness of the Proposed Algorithm

Queriability formula needs the value of the frequency for each concept in the input taxonomy over the dataset. However, it is not possible to find the exact frequencies of concepts without annotating the mentions to their entities in the dataset. Similar to [50], we estimate the concept frequencies by sampling a small subset of randomly selected documents from the dataset.

Table 4 Average MRR for APM, APM-L, LW and $DP_{\epsilon=0.05}$ over T1, T2, T3, T4 and T5. Statistically significant difference between APM and LW, between APM and DP, between DP and LW, between APM-L and LW, and between DP and APM-L are in italic, bold, underline, apostrophe(') and star(*), respectively. B denotes a given budget.

		Uniform Cost				Random Cost				Frequency-based Cost			
	B	APM	APM-L	LW	DP	APM	APM-L	LW	DP	APM	APM-L	LW	DP
T1	0.1	0.187	0.187	0.187	0.187	0.239	0.250	0.250	0.255	0.475	0.475	0.475	0.475
	0.2	0.197	0.220	*0.220*	**0.362***	0.204	0.318	*0.318*	**0.384**	0.475	0.475	0.475	0.475
	0.3	0.221	0.394	*0.394*	**0.406**	0.288	0.390	*0.390*	**0.417***	0.475	0.475	0.475	0.475
	0.4	0.394	0.438	*0.438*	**0.459***	0.357	0.440	*0.440*	**0.460***	0.475	0.475	0.475	0.475
	0.5	0.438	0.492	*0.492*	**0.492**	0.421	0.492	*0.492*	**0.486**	0.475	0.475	0.475	0.475
	0.6	0.492	0.501	0.501	0.492	0.471	0.496	*0.496*	0.488	0.475	0.475	0.475	0.475
	0.7	0.501	0.502	0.502	0.493	0.494	0.502	*0.502*	0.494	0.475	0.475	0.475	0.475
	0.8	0.502	0.507	0.507	0.507	0.502	0.503	0.503	0.502	0.475	0.475	0.475	0.475
	0.9	0.507	0.507	0.507	0.507	0.502	0.506	0.506	0.506	0.475	0.475	0.475	0.475
T2	0.1	0.504	0.479	0.479	0.479	0.481	0.466	0.466	0.476	0.589	0.598	0.589	0.598
	0.2	0.534	0.566	*0.566*	**0.566**	0.564	0.567	0.571'	**0.591***	0.595	0.662	*0.662*	**0.662**
	0.3	0.580	0.629	*0.629*	**0.629**	0.607	0.638	*0.638*	**0.652***	0.667	0.668	0.668	0.671
	0.4	0.651	0.718	*0.718*	**0.729**	0.666	0.686	*0.686*	**0.713***	0.668	0.741	*0.741*	**0.739**
	0.5	0.673	0.745	*0.745*	**0.734**	0.673	0.738*	*0.738*	0.728	0.703	0.747	*0.747*	**0.747**
	0.6	0.746	0.751	0.751	0.750	0.702	0.747	0.748	0.746	0.709	0.748	*0.748*	**0.748**
	0.7	0.751	0.758	0.758	0.763	0.747	0.755	0.755	0.751	0.747	0.748	0.748	0.748
	0.8	0.757	0.764	0.764	0.764	0.755	0.760	0.761	0.759	0.748	0.748	0.748	0.748
	0.9	0.757	0.764	0.764	0.764	0.758	0.763	0.764	0.761	0.748	0.748	0.748	0.756
T3	0.1	0.453	0.469	0.469	0.469	0.458	0.475	0.499'	0.524*	0.407	0.407	0.407	0.447
	0.2	0.453	0.594	*0.594*	**0.594**	0.545	0.608	*0.615'*	**0.629***	0.531	0.531	0.531	0.531
	0.3	0.579	0.679	*0.679*	**0.679**	0.616	0.682	*0.698'*	**0.701***	0.577	0.577	0.577	0.594
	0.4	0.664	0.734	*0.734*	**0.734**	0.648	0.732	*0.732*	**0.734**	0.626	0.679	*0.679*	**0.688***
	0.5	0.719	0.739	*0.739*	**0.739**	0.685	0.738	*0.738*	**0.744**	0.669	0.734	*0.734*	**0.734**
	0.6	0.737	0.760	*0.760*	**0.760**	0.730	0.752	*0.752*	0.751	0.715	0.734	0.734	0.734
	0.7	0.758	0.760	0.760	0.760	0.750	0.760	0.760	0.752	0.676	0.739	0.739	0.739
	0.8	0.760	0.760	0.760	0.760	0.759	0.760	0.760	0.760	0.701	0.739	0.739	0.739
	0.9	0.760	0.760	0.760	0.760	0.759	0.760	0.760	0.760	0.739	0.749	0.749	0.754
T4	0.1	0.343	0.413	*0.413*	**0.413**	0.408	0.423	*0.439'*	**0.426**	0.344	0.344	0.344	**0.363***
	0.2	0.363	0.456	*0.456*	**0.456**	0.422	0.457	*0.467'*	**0.462***	0.364	0.364	0.365	**0.394***
	0.3	0.433	0.491	*0.491*	**0.496***	0.440	0.516	*0.518*	**0.508**	0.389	0.395	0.391	**0.488***
	0.4	0.435	0.563*	*0.563*	**0.544**	0.441	0.540	*0.548'*	**0.547***	0.525	0.556*	*0.556*	0.544
	0.5	0.456	0.573	*0.573*	**0.601***	0.464	0.587	*0.587*	**0.588**	0.544	0.584	*0.581*	**0.584**
	0.6	0.480	0.608	*0.608*	**0.612**	0.503	0.606	*0.606*	**0.597**	0.548	0.609	*0.609*	**0.609**
	0.7	0.547	0.627*	*0.627*	**0.621**	0.539	0.622	*0.622*	**0.621**	0.525	0.609	*0.609*	**0.609**
	0.8	0.550	0.628	*0.628*	**0.627**	0.556	0.627	*0.627*	**0.627**	0.539	0.619	*0.619*	**0.619**
	0.9	0.555	0.629	*0.629*	**0.629**	0.563	0.629	*0.629*	**0.629**	0.559	0.628	*0.628*	**0.628**
T5	0.1	0.376	0.442	*0.442*	**0.448**	0.431	0.471	*0.475'*	**0.467**	0.373	0.374	*0.379'*	**0.379***
	0.2	0.450	0.516	*0.516*	**0.510**	0.458	0.540*	*0.540*	**0.528**	0.394	0.394	*0.409'*	**0.415***
	0.3	0.501	0.571	*0.571*	**0.571**	0.523	0.577	*0.580'*	**0.580***	0.416	0.545*	*0.545*	**0.461**
	0.4	0.543	0.608*	*0.608*	**0.574**	0.561	0.605	*0.605*	**0.600**	0.557	0.608	*0.608*	**0.608**
	0.5	0.572	0.621*	*0.621*	**0.608**	0.581	0.624	*0.624*	**0.617**	0.579	0.624	*0.624*	**0.624**
	0.6	0.607	0.638	*0.638*	**0.624**	0.608	0.637	*0.637*	**0.625**	0.600	0.626	*0.625*	**0.627**
	0.7	0.613	0.647	*0.647*	**0.647**	0.621	0.648*	*0.648*	**0.638**	0.556	0.648	*0.648*	**0.648**
	0.8	0.633	0.653	*0.653*	**0.651**	0.631	0.653	*0.653*	**0.651**	0.581	0.652	*0.652*	**0.652**
	0.9	0.642	0.656	*0.656*	**0.656**	0.643	0.655	*0.655*	**0.655**	0.569	0.655	*0.655*	**0.655**

We compute the frequency of each concept using an estimation error rate of 5% under the 95% confidence level, which is about 400 documents for all datasets. We also smooth the sampled frequencies using Bayesian m-estimates with smoothing parameter of 1 and uniform priors. We denote the LEVEL-WISE algorithm as *LW* and the dynamic programming algorithm as *DP* for brevity. We also compare LW and DP with the *APM* algorithm from [50] which finds a design over a set of concepts. We use all concepts in the taxonomy as a set of concepts for an input to APM. APM uses a scaling technique to convert popularities and costs to positive integers [50]. We set the ϵ value of the scaling for APM to 0.01. As we have mentioned in Section 3, LW uses APM to find the optimal design in each level. We set the scaling factor of APM used by LW to 0.01. We also perform APM algorithm only over the set of leaf concepts in a taxonomy, and denote it as *APM-L*.

Since DP also assumes popularity (u), frequency (d) and cost (w) to be positive integers, we also use scaling to convert the values of popularity, frequency and cost of all concept in the input taxonomy to positive integers [53]. Let $u_{\max}$ be the maximum popularity of all leaf concepts in the taxonomy and $\varepsilon < 1$, we scale $u(C)$ as $\hat{u}(C) = \lfloor \frac{u(C)}{\varepsilon \cdot u_{\max}} \rfloor$. We use similar techniques to scale the values of $d(C)$ and $w(C)$. Intuitively, the smaller the value of ε is, the more exact result DP will deliver. However, DP takes longer to run for smaller values of ε as the range of U, D and $B_{\texttt{total}}$ will become larger. Since we cannot use a very small value of ϵ, our preliminary results of using this scaling technique shows that many concepts have u or d values equals to 0 after scaling. Hence, DP may not explore all feasible designs and may miss some popular concepts whose d value are small. Thus, we modify the aforementioned scaling technique by adding a constant value of 1 to both

$\hat{u}(C)$ and $\hat{d}(C)$. This addition ensures that each concept, after the scaling, contributes in the computation of queriability, and DP will explore all feasible designs. We set the value of ε for DP to 0.05 for the experiments in this section.

Table 4 shows the values of average MRR for APM, APM-L, LW and DP over T1, T2, T3, T4 and T5 for all cost models. Overall, the designs returned by LW and DP improve the effectiveness of answering queries for all taxonomies more than the designs returned by APM. This is because APM does not consider the structural information of the taxonomy. For instance, consider the designs returned by APM, DP, and LW over T5 using uniform cost model. The design of APM includes both *organism* and *person*. The concept *organism* is a parent of *person* and both concepts have almost equally popular in the query workload. LW and DP use structural information of the taxonomy when selecting a design, and thus their selected designs included either *organism* or *person* but not both. LW and DP then spend a remaining budget on other concepts, and improve the effective of answering those queries.

Since APM-L returns designs only over leaf concepts of a taxonomy, as opposed to APM, it does not unnecessarily pick a leaf concept and its ancestor(s) in the same design. Thus, the average MRR values of APM-L is generally higher than the ones of APM. Hence, they are more effective than APM.

LW is generally more effective than APM-L. APM-L and LW chose the same designs over uniform cost model. In uniform cost model, each leaf concept costs equal to its ancestor(s), therefore, LW picks only leaf concepts in its designs. We observe almost the same trend in the results of the frequency-based cost model. Since each leaf node in this cost model is less costly to annotate than its ancestors, as we have explained in Section 1, LW picks mostly leaf concepts in its designs. Nevertheless, when there is not enough budget to select some leaf concepts, e.g., budget 0.1 in T5, LW selects internal nodes. In these cases, APM-L is *not* able to include internal nodes to its designs and delivers significantly lower MRR than LW. The designs returned by APM-L and LW differ more over random cost model compared with other cost models, in particular, for deeper taxonomies, e.g., T3 and T4, and smaller budgets. Shallow taxonomies, e.g., T1 and T2, have only two levels of concepts and the given budgets are usually sufficient to select popular concepts in the leaf level. Hence, both LW and APM-L returns the same designs. Also, the distribution of popularities in the leaf concepts for all taxonomies is quite skewed with a small subset of the leaf concepts being very popular and the rest having very small degree of popularity in the query workload. Hence, using a relatively modest budget, e.g., larger than 0.4, both LW and APM-L can pick all popular concepts in the leaf level. In other cases, because LW is able to pick non-leaf concepts, its results are significantly more effective than the ones of APM-L.

Because DP include both leaf and non-leaf concepts in its designs, it is generally more effective than APM-L. For instance, consider the results of DP and APM-L over T2 for budget 0.3 using random cost model. In this taxonomy, concept *writing* is the parent of both concepts *literary composition* and *dramatic composition*. APM-L picks *literary composition* and *dramatic composition* while DP picks *literary composition* and *writing*. The design of APM-L can effectively answer queries about *literary composition* and *dramatic composition*. The design returned by DP can answer queries about *literary composition* effectively and also provide reasonably effective results for queries about all children of *writing*. Hence, the average MRR values of DP is higher than the ones of APM-L.

The results shown in Table 4 indicate that the designs returned by DP are more effective than those returned by LW for small budgets and small taxonomies such as T1, T2 and T3. However, the designs returned by LW are more effective than those delivered by DP for moderate to large budgets, e.g., 0.4-0.7. Because the frequencies and popularities of concepts in most taxonomies follow a power-law distribution, most of all concepts have frequency and popularity close to zero. When a given budget is very small, neither methods have a sufficient budget to select the concepts with medium or small frequencies and popularities. However, over sufficiently large budgets, both algorithms are able to select some of these concepts. If the DP algorithm does not use sufficiently small values for ϵ, concepts with considerably different frequencies and popularities may end up with equal values of scaled frequencies and popularities. Since it takes a very long time to run DP with an ϵ value less than 0.05, one cannot use a sufficiently small value of ϵ to preserve the differences in popularities and frequencies for all concepts As the DP algorithm is not able to distinguish these concepts, it may not be able to find a design with the most queriability. Since we are able to run LW using sufficiently small scaling factors in its APM component, LW often returns more effective designs than DP over relatively large budgets.

Table 5 shows the values of average MRR for APM, APM-L, and LW over T6, T7, and T8 for all cost models. The results of budgets greater than 0.6 are omitted because they are the same as the budget 0.6. We do *not* report any result for DP over T6, T7 and T8 because the algorithm does not terminate for almost all budgets

Table 5 Average MRR for APM, APM-L and LW over T6 and T8. Statistically significant differences between APM and LW, between APM and APM-L, and between APM-LW and LW are marked in bold, italic and underline, respectively.

		Uniform Cost			Random Cost			Frequency-based Cost		
	B	APM	APM-L	LW	APM	APM-L	LW	APM	APM-L	LW
T6	0.01	0.235	0.235	0.235	*0.375*	0.291	**0.403**	0.259	0.259	0.259
	0.05	0.411	*0.474*	**0.474**	0.412	*0.491*	**0.497**	0.311	*0.327*	**0.327**
	0.1	0.461	*0.564*	**0.564**	0.466	*0.559*	**0.563**	0.333	*0.380*	**0.380**
	0.2	0.583	*0.625*	**0.625**	0.589	*0.624*	**0.625**	0.392	0.397	**0.404**
	0.3	0.627	*0.648*	**0.648**	0.623	*0.647*	**0.648**	0.403	*0.410*	**0.417**
	0.4	0.631	*0.652*	**0.652**	0.638	*0.652*	**0.652**	0.418	*0.589*	**0.589**
	0.5	0.642	0.653	0.653	0.642	*0.653*	**0.653**	0.573	*0.626*	**0.626**
	0.6	0.646	0.653	0.653	0.646	*0.653*	**0.653**	0.577	*0.632*	**0.632**
T7	0.01	0.344	*0.380*	**0.380**	0.385	0.391	**0.406**	0.286	0.285	**0.310**
	0.05	0.448	*0.525*	**0.525**	0.451	*0.517*	**0.530**	0.371	*0.403*	**0.403**
	0.1	0.545	*0.609*	**0.609**	0.550	*0.612*	**0.616**	0.410	*0.460*	**0.460**
	0.2	0.637	*0.669*	**0.669**	0.633	*0.671*	**0.671**	0.464	0.463	0.461
	0.3	0.659	*0.682*	**0.682**	0.667	*0.683*	**0.683**	0.464	*0.602*	**0.602**
	0.4	0.677	0.686	0.686	0.675	*0.686*	**0.686**	0.606	*0.642*	**0.642**
	0.5	0.683	0.686	0.686	0.681	0.686	0.686	0.609	*0.646*	**0.646**
	0.6	0.684	0.686	0.686	0.684	0.686	0.686	0.611	*0.662*	**0.662**
T8	0.001	0.357	0.357	0.357	*0.365*	0.334	**0.407**	0.277	0.277	0.277
	0.005	0.433	*0.452*	**0.452**	0.462	0.470	**0.473**	0.316	*0.336*	**0.334**
	0.01	***0.537***	0.517	0.517	0.535	0.535	0.535	0.355	*0.372*	**0.379**
	0.05	0.686	*0.707*	**0.707**	0.712	0.712	0.712	0.479	*0.561*	**0.561**
	0.1	0.730	*0.738*	**0.738**	0.738	0.738	0.738	0.541	*0.598*	**0.598**
	0.2	0.740	0.743	0.743	0.742	0.742	0.742	0.570	*0.704*	**0.704**
	0.3	0.742	0.743	0.743	0.743	0.743	0.743	0.665	*0.716*	**0.716**
	0.4	0.743	0.743	0.743	0.743	0.743	0.743	0.680	*0.737*	**0.737**
	0.5	0.743	0.743	0.743	0.743	0.743	0.743	0.651	*0.737*	**0.737**
	0.6	0.743	0.743	0.743	0.743	0.743	0.743	0.668	*0.737*	**0.737**

after several days. Table 5 indicates that the designs returned by LW are generally more effective than the ones of APM and APM-L. Similar to the experiments over T1-T5, APM-L and LW return the same designs for uniform cost model. Because of a very skewed distribution of concept popularity in T6, T7 and T8, budgets of 0.4 for T6 and T7, and 0.2 for T8 are sufficient to create a design that includes all leaf concepts that appear in the query workloads in random and frequency-based cost models. Hence, APM, APM-L and LW deliver almost equally effective designs for relatively large budgets.

To further evaluate and compare the aforementioned algorithms for relatively small budgets, we evaluate APM, APM-L and LW using budgets 0.01 and 0.05 for T6, T7 and T8 and budgets 0.001 and 0.005 for T8 as shown in Table 5. LW is significantly more effective than APM-L and APM for these budgets. For relatively small budgets, it is more preferable to pick relatively many nodes in non-leaf levels instead of choosing a rather small number of leaf concepts. The designs of APM-L contain a small number of leaf concepts and significantly improve the effectiveness of answering queries of those concepts. However, the designs of LW contain more concepts, which are the internal nodes in the taxonomy and improve the effectiveness of answering all queries that refer to children of the chosen concepts. Hence, the designs of LW improve the effectiveness of many more queries than the ones of APM-L and APM, therefore, they deliver larger overall MRR.

Table 6 Average MRR of QM and DPC using different values of ϵ over T1, T2 and T3. Statistically significant difference between QM and DPC are marked in bold. B denotes a given budget.

	B	QM	$DPC_{0.05}$	$DPC_{0.1}$	$DPC_{0.2}$
T1	0.25	0.426	**0.299**	**0.364**	0.413
	0.50	0.488	0.464	0.464	0.464
	0.75	0.507	0.507	0.507	0.507
	1	0.507	0.507	0.507	0.507
T2	0.25	0.609	0.596	0.594	0.594
	0.50	0.747	0.743	0.742	**0.695**
	0.75	0.763	0.759	0.755	0.750
	1	0.764	0.764	0.764	0.764
T3	0.25	0.707	0.662	**0.655**	**0.655**
	0.50	0.756	0.747	0.741	**0.729**
	0.75	0.760	0.760	0.758	0.749
	1	0.760	0.760	0.760	0.760

8.3.1 Dynamic Programming with Cost-Dependency

Table 6 show the values of MRR for Queriability Maximization (QM) and dynamic programming algorithm with cost dependencies (*DPC*) over T1, T2 and T3 taxonomies. The cost for each concept in these taxonomies has been randomly generated and depends on which ancestors of the concept have been selected in the design. The budgets that cover all leaf nodes in T1, T2 and T3 are 1, and the budget that cover all nodes are 1.25, 1.1 and 1.1, respectively. QM is a brute force algorithm that explores all feasible solutions and finds the design with maximum queriability. Because the space of the possible solutions for this problem is larger than the original CECD problem, it takes much longer to run QM for this problem. Hence, we have covered a smaller range of budgets in this set of experiments. The results

Table 7 Average running time (minute) of APM, LW, DP and DPC.

	APM	LW	DP		DPC	
			$\epsilon = 0.05$	0.1	$\epsilon = 0.1$	0.2
T4	1	1	127	11	151	12
T5	1	1	184	15	778	77
T6	1	1	-	-	-	520
T7	1	1	-	-	-	-
T8	1	1	-	-	-	-

shown in Table 6 indicate that the smaller the value of ϵ, the closer the average MRR of the designs returned by DPC are to the ones delivered by QM.

8.4 Efficiency of Proposed Algorithms

We measure the running times of LW and DP over moderate and large taxonomies, i.e., T4, T5, T6, T7 and T8, and set the available main memory of Java Virtual Machine to 64GB. Table 7 shows the average running times of APM, LW and DP for T4, T5, T6, T7 and T8 over budgets 0.1 to 0.9. Some results of DP are not reported because the algorithms did not finish after a day. Overall, LW is as efficient as APM, and it is more efficient than DP. Because the size of the table required in the DP algorithm is substantially large for $\epsilon = 0.05$, it occupies most of the available main memory. Thus, the running time of DP is longer than APM and LW. Therefore, LW scales for large taxonomy and is efficient for a design-time task. On the other hand, DP has a reasonable running time for T4 and T5, but it does not scale for large taxonomy such as T6, T7 and T8.

8.4.1 Dynamic Programming with Cost-Dependency

Table 7 shows the average running times of DPC for T4, T5, T6, T7 and T8 over budgets 0.25, 0.5, 0.75 and 1 using the scaling factor, ϵ, of 0.1 and 0.2. We do not report the running time of DPC using $\epsilon = 0.1$ for T6, T7 and T8 and DPC using $\epsilon = 0.2$ for T7 and T8 because the algorithm did not finish after a day. Because DPC requires a larger table than the one required for DP, the running time of DPC is longer than that of DP for the same scaling factor, i.e., ϵ. Hence, we run DPC using only ϵ values of 0.1 and 0.2. Overall, DPC with $\epsilon = 0.1$ is reasonably efficient to perform a design-time task for a taxonomy of up to size 70, and DPC with $\epsilon = 0.2$ is reasonably efficient for a taxonomy of up to size 200.

Table 8 The numbers of queries with multiple concepts with the minimum, average and maximum numbers of concepts per query for T4, T5, T6, T7 and T8.

Taxonomy	T4	T5	T6	T7	T8
#queries	687	1578	1403	1882	2603
minimum #concepts	2	2	2	2	2
maximum #concepts	4	4	5	5	5
average #concepts	2.2	2.1	2.2	2.2	2.2

8.5 Queries With Multiple Concepts

8.5.1 Validation and Effectiveness

We have selected all queries with multiple concepts which belong to T1, T2 and T3 from our query workload and filtered out the queries whose ranking quality is not improved by annotating all concepts in the corresponding taxonomy. This results in 6, 37 and 8 queries over T1, T2 and T3, respectively. The number of concepts for each query is 2. Because there are not enough queries with multiple concepts for T1 and T3, we do not evaluate our models over T1 and T3. Since the number of queries with one concept over T2 is seven times larger than the number of multiple concepts, we randomly select a subset of queries with one concept from the original query workload and combine them with the queries with multiple concepts over T2. The new query workload contains 106 queries. We run Oracle, the queriability maximization over queries with multiple concepts (MQM), the Level-wise algorithm for multiple concepts (MLW), and the Level-wise algorithm (LW) over the query workload. Oracle is described in Section 8.2. MQM enumerates all feasible designs over the input taxonomy and returns the one with maximum queriability as computed in Section 6. Table 9 shows the values of average MRR for Oracle, MQM, MLW and LW algorithms over T2. MQM generally returns the designs similar to those of Oracle. The designs returned by MLW also deliver close average MRR the ones delivered by MQM in most cases. Overall, MLW significantly outperforms LW over T2.

We have also evaluated the effectiveness of MLW and LW over taxonomies T4, T5, T6, T7 and T8. We have selected all queries with multiple concepts over T4, T5, T6, T7 and T8 from our query workload. Table 8 shows the numbers of queries with multiple concepts and the minimum, average and maximum numbers of concepts per query for T4, T5, T6, T7 and T8. Table 10 shows the values of MRR for MLW and LW algorithms over T4, T5, T6, T7 and T8. Overall, the designs returned by MLW deliver significantly higher MRR than the designs selected by LW over all taxonomies.

Table 9 Average MRR for Oracle, MQM, MLW and LW over T2. Statistically significant difference between MQM and Oracle, MQM and MLW, and MLW and LW are marked in italic, bold, and underline, respectively. B denotes a given budget.

	Uniform Cost				Random Cost				Frequency-based Cost			
B	Oracle	MQM	MLW	LW	Oracle	MQM	MLW	LW	Oracle	MQM	MLW	LW
0.1	*0.517*	0.430	0.491	0.491	*0.577*	0.491	0.465	0.454	0.569	0.569	0.569	0.569
0.2	0.593	**0.577**	0.516	0.523	0.651	0.596	0.569	0.529	0.657	0.657	0.657	0.657
0.3	*0.747*	0.602	0.600	0.526	*0.785*	0.657	0.670	0.577	*0.758*	0.682	0.682	0.682
0.4	0.826	0.796	0.796	0.563	0.826	0.791	0.791	0.618	0.829	0.829	0.829	0.829
0.5	0.850	0.821	0.821	0.576	0.846	0.832	0.832	0.626	0.832	0.832	0.832	0.829
0.6	0.859	0.852	0.852	0.576	0.862	0.837	0.837	0.663	0.832	0.832	0.832	0.832
0.7	0.865	0.865	0.865	0.823	0.862	0.852	0.852	0.768	0.832	0.832	0.832	0.832
0.8	0.865	0.865	0.865	0.832	0.862	0.862	0.862	0.835	0.832	0.832	0.832	0.832
0.9	0.865	0.865	0.865	0.865	0.862	0.862	0.862	0.858	0.861	0.861	0.861	0.832

We have observed less difference between the results of MLW and LW when the distribution of concept popularity in the taxonomy is less skewed. This is because, when the popularity distribution is very skewed, both algorithms select all or most relatively popular concepts. Hence, they selected designs are almost equally effective. For example, the distribution of concept frequencies in T6 is considerably more skewed than those of the concepts in T7, thus, the designs delivered by MLW and LW over T6 are significantly more different than the ones these algorithms return over T7.

8.5.2 Efficiency

We measure the average running times of MLW over moderate and large taxonomies, i.e., T4, T5, T6, T7 and T8, and set the available main memory of the Java Virtual Machine to 64GB. The average running times of MLW for T4, T5, T6, T7 and T8 over budgets 0.1 to 0.9 are 3, 3, 4, 4 and 8 minutes, respectively, which are reasonable for a design-time task.

9 Related Work

Researchers have examined the problem of selecting cost effective designs from an unorganized set of concepts for annotation [50]. Nevertheless, the concepts in most real-world domains are maintained in taxonomies rather than unorganized sets. As the framework proposed in [50] 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 2.4 that because the algorithms in [50] 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 8.3 also indicate that the algorithms in [50] deliver considerably less effective designs than the ones that take the structure of the taxonomy into account. Furthermore, the authors in [50] do not consider the cost dependencies between concepts. They also assume that each query refers to only a single concept.

There is a large body of work on building large-scale data management systems for annotating and extracting entities and relationships from unstructured data

Table 10 Average MRR for MLW and LW over T4, T5, T6, T7 and T8. Statistically significant differences between MLW and LW are marked in bold. B denotes a given budget.

		Uniform Cost		Random Cost		Frequency-based Cost	
	B	MLW	LW	MLW	LW	MLW	LW
T4	0.1	**0.334**	0.274	**0.448**	0.368	**0.574**	0.487
	0.2	**0.622**	0.488	**0.638**	0.487	**0.721**	0.629
	0.3	**0.748**	0.509	**0.827**	0.601	0.770	0.770
	0.4	**0.873**	0.712	**0.880**	0.712	0.773	0.784
	0.5	**0.873**	0.733	**0.887**	0.749	0.815	0.799
	0.6	0.873	0.833	**0.890**	0.813	0.845	0.799
	0.7	0.901	0.872	0.901	0.868	0.845	0.862
	0.8	0.901	0.862	0.912	0.894	0.902	0.872
	0.9	0.929	0.930	0.924	0.930	0.900	0.873
T5	0.1	**0.466**	0.430	**0.619**	0.466	**0.674**	0.536
	0.2	**0.799**	0.673	**0.803**	0.645	0.708	0.695
	0.3	0.807	0.810	**0.819**	0.729	0.719	0.728
	0.4	**0.826**	0.743	**0.828**	0.784	**0.827**	0.745
	0.5	0.834	0.811	0.833	0.819	**0.838**	0.773
	0.6	0.848	0.848	0.866	0.845	**0.852**	0.800
	0.7	0.883	0.840	0.883	0.840	0.863	0.851
	0.8	0.883	0.846	0.883	0.850	0.874	0.856
	0.9	0.883	0.869	0.883	0.862	0.883	0.876
T6	0.1	**0.794**	0.473	**0.796**	0.545	0.765	0.763
	0.2	**0.813**	0.508	**0.840**	0.561	**0.827**	0.777
	0.3	**0.883**	0.667	**0.882**	0.690	**0.853**	0.799
	0.4	**0.884**	0.795	**0.884**	0.810	**0.857**	0.813
	0.5	0.890	0.887	0.890	0.887	**0.871**	0.820
	0.6	0.892	**0.920**	0.894	0.897	0.888	0.887
	0.7	0.901	0.838	**0.912**	0.833	**0.908**	0.896
	0.8	0.932	0.906	0.932	0.913	**0.932**	0.903
	0.9	0.932	0.906	0.932	0.913	0.939	0.926
T7	0.1	0.764	0.778	0.771	0.778	**0.723**	0.605
	0.2	**0.858**	0.805	**0.856**	0.552	**0.762**	0.717
	0.3	0.860	0.848	0.860	0.852	**0.779**	0.731
	0.4	0.872	0.876	0.872	0.876	**0.826**	0.791
	0.5	**0.875**	0.768	**0.876**	0.767	0.845	0.846
	0.6	0.880	0.867	0.882	0.861	0.855	0.855
	0.7	0.889	0.895	0.889	0.895	0.866	0.866
	0.8	0.889	0.895	0.889	0.845	0.875	0.874
	0.9	0.890	0.896	0.901	0.896	0.895	0.895
T8	0.1	**0.882**	0.783	**0.880**	0.793	**0.776**	0.548
	0.2	**0.883**	0.848	**0.890**	0.844	**0.785**	0.511
	0.3	**0.887**	0.848	**0.893**	0.859	**0.791**	0.589
	0.4	**0.889**	0.867	0.895	0.876	**0.798**	0.654
	0.5	**0.894**	0.887	0.896	0.888	**0.802**	0.736
	0.6	0.896	0.896	0.898	0.892	**0.846**	0.750
	0.7	0.896	0.896	0.900	0.892	**0.881**	0.754
	0.8	0.901	0.901	0.901	0.892	0.891	0.894
	0.9	0.901	0.901	0.901	0.894	0.900	0.900

sources [17, 31]. In particular, researchers have proposed several techniques to optimize the running time and/or required computational resources 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 [30, 31]. 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. Furthermore, our framework can handle other types of costs than computational overheads.

We have previously published our preliminary results on the problem of cost-effective conceptual design over taxonomies in a short workshop paper [52]. In that paper, we have formally defined the problem for tree taxonomies and introduced an approximation algorithm for it. The current submission significantly extends the previous work in the following directions. First, we present an exact algorithm to solve the problem over tree taxonomies. Second, the cost of annotating a concept may change if the instances of its ancestor(s) in the taxonomy have already been annotated. We propose an exact algorithm for this case of CECD problem. Third, queries over a dataset often refer to entities from multiple concepts. We introduce the problem of CECD in which queries may refer to multiple concepts and propose efficient algorithms to solve the problem over both unorganized sets of concepts and tree taxonomies. Fourth, we formalize and investigate the problem of CECD over taxonomies that are directed acyclic graphs. Fifth, the current submission contains extensive empirical results that validate our formalizations of the aforementioned variations of the problem and evaluate the effectiveness and efficiency of our proposed algorithms. Finally, this submission contains the proofs for our theoretical results in [52].

10 Conclusion

We introduced the problem of cost-effective conceptual design using taxonomies, where given a taxonomy, one would like to find a subset of concepts in the taxonomy whose total cost does not exceed a given budget and improves the effectiveness of answering queries the most. We proved the problem is NP-hard and proposed efficient approximation algorithms and exact algorithm with pseudo-polynomial running time for different versions of the problem over tree taxonomies. We also proved that it is not possible to find any approximation algorithm with reasonably small approximation ratio or pseudo-polynomial time exact algorithm for the problem when the taxonomy is a directed acyclic graph.

References

1. Abiteboul S, Manolescu I, Rigaux P, Rousset M, Senellart P (2011) Web Data Management. Cambridge University Press
2. Anderson M, Cafarella M, Jiang Y, Wang G, Zhang B (2014) An Integrated Development Environment for Faster Feature Engineering. PVLDB 7(13):1657–1660
3. Anderson M, et al (2013) Brainwash: A data system for feature engineering. In: CIDR
4. Arora S, Manokaran R, Moshkovitz D, Weinstein O (2011) Inapproximability of densest κ-subgraph from average-case hardness. people.csail.mit.edu/dmoshkov/papers
5. Arulselvan A (2014) A note on the set union knapsack problem. Discrete Applied Mathematics 169:214–218
6. Bhaskara A, Charikar M, Vijayaraghavan A, Guruswami V, Zhou Y (2012) Polynomial Integrality Gaps for Strong SDP Relaxations of Densest K-subgraph. In: SODA, pp 388–405
7. Boehm B, et al (2000) Software development cost estimation approaches: A survey. Annals of Software Engineering 10(1-4):177–205
8. Chakrabarti S, Puniyani K, Das S (2007) Optimizing Scoring Functions and Indexes for Proximity Search in Type-annotated Corpora. In: WWW, pp 717–726
9. Chang CH, Kayed M, Girgis MR, Shaalan KF (2006) A survey of web information extraction systems. TKDE 18:1411–1428
10. Chiticariu L, Krishnamurthy R, Li Y, Raghavan S, Reiss FR, Vaithyanathan S (2010) Systemt: An algebraic approach to declarative information extraction. In: ACL, pp 128–137
11. Chiticariu L, Li Y, Raghavan S, Reiss FR (2010) Enterprise information extraction: Recent developments and open challenges. In: SIGMOD, pp 1257–1258
12. Chiticariu L, Li Y, Reiss FR (2013) Rule-based information extraction is dead! long live rule-based information extraction systems! In: EMNLP, pp 827–832
13. Chu-Carroll J, et al (2006) Semantic Search via XML Fragments: a High-Precision Approach to IR. In: SIGIR, pp 445–452
14. Demidova E, Zhou X, Oelze I, Nejdl W (2010) Evaluating evidences for keyword query disambiguation in entity centric database search. In: DEXA, pp 240–247
15. Deshpande O, Lamba D, Tourn M, Das S, Subramaniam S, Rajaraman A, Harinarayan V, Doan

A (2013) Building, Maintaining, and Using Knowledge Bases: A Report from the Trenches. In: SIGMOD, pp 1209–1220
16. Dill S, et al (2003) Semtag and seeker: Bootstrapping the semantic web via automated semantic annotation. In: WWW, pp 178–186
17. Doan A, Ramakrishnan R, Vaithyanathan S (2006) Managing information extraction: state of the art and research directions. In: SIGMOD, pp 799–800
18. Doan A, Naughton JF, Ramakrishnan R, Baid A, Chai X, Chen F, Chen T, Chu E, DeRose P, Gao BJ, Gokhale C, Huang J, Shen W, Vuong B (2008) Information extraction challenges in managing unstructured data. SIGMOD Record 37(4):14–20
19. Dong XL, Saha B, Srivastava D (2013) Less is More: Selecting Sources Wisely for Integration. PVLDB 6(2):37–48
20. Downey D, Etzioni O, Soderland S (2005) A probabilistic model of redundancy in information extraction. In: IJCAI
21. Fagin R, Kimelfeld B, Reiss F, Vansummeren S (2013) Spanners: A formal framework for information extraction. In: PODS, pp 37–48
22. Furche T, Guo J, Maneth S, Schallhart C (2016) Robust and noise resistant wrapper induction. In: SIGMOD, pp 773–784
23. GarciaMolina H, Ullman J, Widom J (2008) Database Systems: The Complete Book. Prentice Hall
24. Gulhane P, et al (2011) Web-scale information extraction with vertex. In: ICDE, pp 1209–1220
25. Gupta S, Manning CD (2014) Improved pattern learning for bootstrapped entity extraction. In: CoNLL, pp 98–108
26. Gupta S, MacLean DL, Heer J, Manning CD (2014) Research and applications: Induced lexico-syntactic patterns improve information extraction from online medical forums. JAMIA 21(5):902–909
27. Hua W, Wang Z, Wang H, Zheng K, Zhou X (2015) Short text understanding through lexical-semantic analysis. In: ICDE, pp 495–506
28. Huang J, Yu C (2010) Prioritization of Domain-Specific Web Information Extraction. In: AAAI
29. Isozaki H, Kazawa H (2002) Efficient support vector classifiers for named entity recognition. In: COLING, pp 1–7
30. Jain A, Doan A, Gravano L (2008) Optimizing SQL Queries over Text Databases. In: ICDE
31. Kanani P, et al (2012) Selecting actions for resource-bounded information extraction using reinforcement learning. In: WSDM, pp 253–262
32. Khot S (2004) Ruling out PTAS for Graph Minbisection, Densest Subgraph and Bipartite Clique. In: FOCS, pp 136–145
33. Kimelfeld B (2014) Database principles in information extraction. In: PODS, pp 156–163
34. Liu K, et al (2015) Meshlabeler: improving the accuracy of large-scale mesh indexing by integrating diverse evidence. Bioinformatics 31(13):i339–i347
35. Manning CD, Raghavan P, Schütze H, et al (2008) An Introduction to Information Retrieval. Cambridge University Press
36. Manurangsi P (2016) Almost-polynomial ratio ETH-hardness of approximating densest k-subgraph. CoRR abs/1611.05991, URL `http://arxiv.org/abs/1611.05991`
37. McCallum A (2005) Information Extraction: Distilling Structured Data From Unstructured Text. ACM Queue pp 48–57
38. Mintz M, Bills S, Snow R, Jurafsky D (2009) Distant supervision for relation extraction without labeled data. In: ACL, pp 1003–1011
39. Mork J, Demner-Fushman D, Schmidt S, Aronson A (2014) Recent enhancements to the nlm medical text indexer. In: CLEF (Working Notes), pp 1328–1336
40. Nallapati R, Manning CD (2008) Legal docket-entry classification: Where machine learning stumbles. In: EMNLP, pp 438–446
41. Pound J, Ilyas I, Weddell G (2010) Expressive and Flexible Access to Web-Extracted Data: A Keyword-based Structured Query Language. In: SIGMOD, pp 423–434
42. Ratner AJ, De Sa CM, Wu S, Selsam D, Ré C (2016) Data programming: Creating large training sets, quickly. In: NIPS, pp 3567–3575
43. Rekatsinas T, Dong XL, Srivastava D (2014) Characterizing and selecting fresh data sources. In: SIGMOD, pp 919–930
44. Sanderson M (2008) Ambiguous queries: Test collections need more sense. In: SIGIR, pp 499–506
45. Sarawagi S (2008) Information extraction. Foundations and Trends® in Databases 1:261–377
46. Satpal S, Bhadra S, Sellamanickam S, Rastogi R, Sen P (2011) Web information extraction using markov logic networks. In: KDD, pp 1406–1414
47. Shen W, Doan A, Naughton JF, Ramakrishnan R (2007) Declarative information extraction using datalog with embedded extraction predicates. PVLDB pp 1033–1044
48. Shen W, DeRose P, McCann R, Doan A, Ramakrishnan R (2008) Toward best-effort information extraction. In: SIGMOD, pp 1031–1042
49. Suchanek F, et al (2007) Yago: A core of semantic knowledge unifying wordnet and wikipedia. In: WWW, pp 697–706

50. Termehchy A, Vakilian A, Chodpathumwan Y, Winslett M (2014) Which concepts are worth extracting? In: SIGMOD, pp 779–790
51. Vakilian A, Chodpathumwan Y, Termehchy A, Nayyeri A (2015) Cost-effective conceptual design using taxonomies. CoRR abs/1503.05656, URL http://arxiv.org/abs/1503.05656, 1503.05656
52. Vakilian A, Chodpathumwan Y, Termehchy A, Nayyeri A (2017) Cost-effective conceptual design over taxonomies. In: WebDB, pp 35–40
53. Vazirani V (2001) Approximation Algorithms. Springer
54. Wang DZ, Franklin MJ, Garofalakis M, Hellerstein JM, Wick ML (2011) Hybrid in-database inference for declarative information extraction. In: SIGMOD, pp 517–528
55. Wu W, Li H, Wang H, Zhu K (2012) Probase: A probabilistic taxonomy for text understanding. In: SIGMOD, pp 481–492

Appendix

A Proofs

Proof of Theorem 1

Proof The problem of CECD can be reduced to the problem of choosing a cost-effective design from a set of concepts by creating a taxonomy $\mathcal{X} = (R, \mathcal{C}, \mathcal{R})$ where all nodes except for R are leaf concepts. Since the problem of choosing cost-effective concepts from a set of concepts is **NP**-hard [50], CECD is also **NP**-hard. □

Proof of Lemma 1

Proof We have $\sum_{C \in \texttt{free}(\mathcal{S})} u(C)d(C) \leq u(C_{\max}) \sum_{C \in \texttt{free}(\mathcal{S})} d(C)$. Since the frequencies of leaf concepts in $\mathcal{X}$ follow a power-law distribution, we have that $\sum_{C \in \texttt{leaf}(\mathcal{C})} d(C) \leq 1 + \log(|\texttt{leaf}(\mathcal{C})|)$ where $\texttt{leaf}(\mathcal{C})$ is the set leaf concepts in $\mathcal{C}$ and $|\texttt{leaf}(\mathcal{C})|$ is the number of such concepts. Since $|\texttt{leaf}(\mathcal{C})| \leq |\mathcal{C}|$, we have that $QU(\texttt{free}(\mathcal{S})) \leq \sum_{C \in \texttt{free}(\mathcal{S})} u(C)d(C) \leq (1 + \log|\mathcal{C}|)\, u(C_{\max}) \leq 2u(\mathcal{C}_{\max}) \log|\mathcal{C}|$. □

Proof of Theorem 2

Proof Let $\mathcal{S}^*$ be a disjoint design over $\mathcal{X}$ with total cost at most B that maximizes QU function. Let $\mathcal{S}^*[i]$ be the set of concepts in S^* of level i, i.e., $\mathcal{S}^*[i] = \mathcal{S}^* \cap \mathcal{C}[i]$. Since $\mathcal{S}^*$ is a disjoint design, for all $1 \leq i, j \leq h$, $\texttt{part}(\mathcal{S}^*[i]) \cap \texttt{part}(\mathcal{S}^*[j]) = \emptyset$. Thus, $QU(\mathcal{S}^*) = \sum_{1 \leq i \leq h} QU(\mathcal{S}^*[i]) + QU(\texttt{free}(\mathcal{S}^*))$. Next, we consider the two possible cases. First, assume that $\sum_{i=1}^{h} QU(\mathcal{S}^*[i]) \geq QU(\texttt{free}(\mathcal{S}^*))$. It follows that the Level-wise algorithm returns a $(\frac{2h}{\texttt{pr}_{\min}})$-approximate solution of the disjoint CECD defined over $\mathcal{X}$.

In the other case in which $QU(\texttt{free}(\mathcal{S}^*))$ is larger than the other term, by Lemma 1, extracting the concept with the maximum u value gives a $(\frac{4\log(|\mathcal{C}|)}{\texttt{pr}_{\min}})$-approximation. These two cases together imply that we have an $O(\frac{h+\log|\mathcal{C}|}{\texttt{pr}_{\min}})$-approximation. □

Proof of Theorem 4

Proof Construct an instance $\mathcal{M}$ of SU-Knapsack from the MC-CECD instance as described in the preceding paragraph of Theorem 4.

For each pair of concepts C_i, C_j, the total cost of $\mathcal{I}_1 := (\{C_i\}, \{C_j\})$ is equal to the total cost of $\mathcal{I}_2 := (\{C_i\}, \{C_j\}, \{C_i, C_j\})$, and all elements have positive profits. If items $\{C_i\}, \{C_j\}$ are selected in an optimal solution of $\mathcal{M}$, item $\{C_i, C_j\}$ will also be picked. Thus, an optimal solution of $\mathcal{M}$ corresponds to a design $\mathcal{S}$ in MC-CECD: $\mathcal{I}_{\mathcal{S}} = \emptyset \cup \bigcup_{C \in \mathcal{S}} \{C\} \cup \bigcup_{C_1, C_2 \in \mathcal{S}} \{C_1, C_2\}$. Furthermore, the profit of $\mathcal{I}_{\mathcal{S}}$ is: $p(\mathcal{I}_{\mathcal{S}}) = p(\emptyset) + \sum_{C \in \mathcal{S}} p(\{C\}) + \sum_{C_1, C_2 \in \mathcal{S}} p(\{C_1, C_2\}) = QU(\mathcal{S})$. Hence, the algorithm of [5] return a design whose queribility is at least $(1 - 1/e^{\frac{1}{k+1}})$ time the queribility of an optimal design. □

Proof of Theorem 5

Proof We use the following construction to reduce Densest-k-Subgraph to CECD problem over DAG. Given a graph $G = (V, E)$ and a number k, we build an instance of the CECD over a DAG taxonomy as follows. For each edge $e \in E$, we introduce a leaf concept a_e, and for each vertex $v \in V$, we introduce a leaf concept a_v and a non-leaf concept S_v such that S_v is an ancestor of a_v and all a_e corresponding to the incident edges of v in G. Further, we set the budget B to k, the cost of each non-leaf concept to 1, and the cost of leaf concepts to $k + 1$. If we select S_v and S_u in the design and $(u, v) \in E$, a_e will be a singleton partition. We set the popularities and frequencies of all concepts in the taxonomy respectively to the same fixed values u and d. Let m be the number of edges in G and n be the number of vertices in G. For each partition $p \in \texttt{part}(\mathcal{S})$, we set $d(p) = 1/\beta$ if p is a singleton edge concept and $d(p) = 1$ otherwise. β is a parameter which we determine later in the proof. Since leaf concepts are not affordable, there is an optimal design with exactly k non-leaf concepts. In each design $\mathcal{S}$ of size k, the contribution of every leaf concept in a non-singleton edge concept partition is exactly ud. Let $H_{\mathcal{S}}$ be the set of vertices in G whose corresponding non-leaf concepts in $\mathcal{C}$ are in $\mathcal{S}$. $E(H_{\mathcal{S}})$ denotes the set of edges with both endpoints in $H_{\mathcal{S}}$. It corresponds to the set of edge-concepts of $\mathcal{C}$ whose both non-leaf concepts associated with their endpoints are in $\mathcal{S}$. Let $\mathcal{S}_{\text{OPT}}$ be the solution of CECD corresponding to an optimal solution of the Densest-k-Subgraph. We have $QU(\mathcal{S}_{\text{OPT}}) = ud(\beta \cdot t + m + n - t)$ where t denotes the number of edges in $H_{\mathcal{S}_{\text{OPT}}}$. Let $\mathcal{A}$ be an α-approximation algorithm of CECD, and $\mathcal{S}_{\mathcal{A}}$ be the α-approximate design returned by $\mathcal{A}$. Let $QU(\mathcal{S}_{\mathcal{A}}) = ud(t' \cdot \beta + m + n - t')$ where t' is the number of edges whose both endpoint are in $\mathcal{S}_{\mathcal{A}}$. Since $\mathcal{A}$ is an α-approximation algorithm of CECD, $t \cdot \beta + m + n - t \leq \alpha(t' \cdot \beta + m + n - t')$. Thus, $t \leq t' \cdot \alpha + (m+n) \cdot \frac{\alpha - 1}{\beta - 1}$. Setting $\beta = (m+n)(\alpha - 1) + 1$ leads to $O(\alpha)$-approximation for the Densest-k-Subgraph. □

Proof of Corollary 1

Proof By setting $\beta = \frac{(m+n)(\alpha-1)}{\epsilon} + 1$ in proof of Theorem 5 and using the hardness result of [32], no polynomial time approximation scheme algorithm exists for CECD unless $\textbf{NP} \subseteq \cap_{\epsilon > 0} \textbf{BPTIME}(2^{n^{\epsilon}})$. □

Proof of Corollary 2

Proof It follows from the hardness result of [36] and Theorem 5. □