Cost-Effective Conceptual Design Using Taxonomies

Ali Vakilian vakilian@mit.edu1, Yodsawalai Chodpathumwan ychodpa@illinois.edu2, Arash Termehchy termehca@oregonstate.edu3 and Amir Nayyeri nayyeria@oregonstate.edu3

Cost-Effective Conceptual Design Using Taxonomies

Abstract It is known that annotating named entities in unstructured and semi-structured data sets by their concepts improves the effectiveness of answering queries over these data sets. Ideally, one would like to annotate entities of all concepts in a given domain in a data set, however, it takes substantial time and computational resources to do so over a large data set. As every enterprise has a limited budget of time or computational resources, it has to annotate a subset of concepts in a given domain whose costs of annotation do not exceed the budget. We call such a subset of concepts a conceptual design for the annotated data set. We focus on finding a conceptual design that provides the most effective answers to queries over the annotated data set, i.e., a cost-effective conceptual design. Since, it is often less time-consuming and costly to annotate small number of general concepts, such as person, than a large number of specific concepts, such as politician and artist, we use information on superclass/subclass relationships between concepts in taxonomies to find a cost-effective conceptual design. We quantify the amount by which a conceptual design with concepts from a taxonomy improves the effectiveness of answering queries over an annotated data set. If the taxonomy is a tree, we prove that the problem is NP-hard and propose an efficient approximation algorithm and an exact pseudo-polynomial time algorithm for the problem. We further prove that if the taxonomy is a directed acyclic graph, given some generally accepted hypothesis, it is not possible to find any approximation algorithm with reasonably small approximation ratio or a pseudo-polynomial algorithm for the problem. Our empirical study using real-world data sets, taxonomies, and query workloads shows that our framework effectively quantifies the amount by which a conceptual design improves the effectiveness of answering queries. It also indicates that our algorithms are efficient for a design-time task with pseudo-polynomial algorithm being generally more effective than the approximation algorithm.

1 Introduction

1.1 Concept Annotation

Unstructured and semi-structured data sets, such as HTML documents, contain enormous information about named entities like people and products [9][12]. Users normally explore these data sets using keyword queries to find information about their entities of interest. Unfortunately, as keyword queries are generally ambiguous, query interfaces may not return the relevant answers for these queries. For example, consider the excerpts of the Wikipedia (wikipedia.org) articles in Figure 1. Assume that a user likes to find information about John Adams, the politician, over this data set. If she submits query $Q_1$: John Adams, the query interface may return the articles about John Adams, the artist, or John Adams, the school, as relevant answers. Users can further disambiguate their queries by adding appropriate keywords. Nonetheless, it is not easy to find such keywords [31]. For instance, if one refines $Q_1$ to John Adams Ohio, the query interface may return the article about John Adams, the high school, as the answer. It will not help either to add keyword Congressman to $Q_1$ as this keyword does not appear in the article about John Adams, the politician. Formulating the appropriate keyword query requires some knowledge about the sought after entity and the data that most users do not usually possess.

To make querying unstructured and semi-structured data sets easier, data management researchers have proposed methods to identify the mentions to entities in these data sets and annotate them by their concepts [9][12]. Figure 2

Figure 1: Wikipedia article excerpts
showed excerpts of the annotated Wikipedia articles whose original versions are shown in Figure 1. Because entities in an annotated data set are disambiguated by their concepts, the query interface can answer queries over these data sets more effectively. Moreover, as the list of concepts used to annotate the data sets are available to users, they can further clarify their queries by mentioning the concepts of entities in these queries. For example, a user who would like to retrieve article(s) about John Adams, the politician, over the annotated Wikipedia data set in Figure 2 may mention the concept of politician in her query. The set of annotated concepts in a data set is the conceptual design for the data set 27. For example, the conceptual design of the data fragment in Figure 2 is $D_1 = \{\text{politician, legislature, artist, school, state, city}\}$. Using $D_1$, the query interface is able to disambiguate all entities in this data fragment.

1.2 Costs of Concept Annotation

Ideally, an enterprise would like to annotate all relevant concepts from a data set to answer all queries effectively. Nonetheless, an enterprise has to spend significant time, financial and computational resources, and manual labor to accurately extract entities of a concept in a large data set [4, 16, 9, 27, 19, 26, 17, 20]. An enterprise usually has to develop or obtain a complex program called concept annotator to annotate entities of a concept from a collection of documents 23. Enterprises develop concept annotator using rule-based or machine learning approaches. In the rule-based approach, developers have to design and write hand-tuned programming rules to identify and annotate entities of a given concept. For example, one rule to annotate entities of concept person is that they start with a capital letter. It is not uncommon for a rule-based concept annotator to have thousands of programming rules, which takes a great deal of resources to design, write, and debug 23.

One may also use machine learning algorithms to develop an extractor for a concept 23. In this approach, developers have to find a set of relevant features for the learning algorithm. Unfortunately, as the specifications of relevant features are usually unclear, developers have to find the relevant features through a time-consuming and labor-intensive process [4, 3]. First, they have to inspect the data set to find some candidate features. For each candidate feature, developers have to write a program to extract the value(s) of the feature from the data set. Finally, they have to train and test the concept annotator using the set of selected features. If the concept annotator is not sufficiently accurate, developers have to explore the data set for new features. As a concept annotator normally uses hundreds of features, developers have to iterate these steps many times to find a set of reasonably effective features, where each iteration usually takes considerable amount of time [4, 3]. The overheads feature engineering and computation have been well recognized in machine learning community 29. Moreover, if concept annotators use supervised learning algorithms, developers have to collect or 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 have to slog through the data set to find relevant features for concept annotators in these domains [4].

Unfortunately, the overheads of developing a concept annotator are not one-time costs. Because the structure and content of underlying data sets evolve over time, annotators should be regularly rewritten and repaired [16]. Recent studies show that many concept annotator need to be rewritten in average about every two months [16]. Thus, the enterprise often have to repeat the resource-intensive steps of developing a concept annotator to maintain an up-to-date annotated data set.

After developing concept annotators, the enterprise executes them over the data set to generate the annotated collection. As most concept annotators perform complex text analysis, such as deep natural language parsing, it may take them days to process a large data set 19, 26, 17, 20. As the content of the data set evolves, extractors should be often rerun to create an updated annotated collection.

1.3 Cost-Effective Conceptual Design

Because the available financial or computational resources of an enterprise are limited, it may not afford to develop, deploy, and maintain annotators for all concepts in a domain. Also, many users may need an annotated data set quickly and cannot wait days for an (updated) annotated collection 26, 19. For example, a reporter who pursues some breaking
news, a stock broker that studies the relevant news and documents about companies, 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. Hence, the enterprise may afford to annotate only a subset of concepts in a domain.

Concepts in many domains are organized in taxonomies. Figure 3 depicts fragments of DBPedia dbpedia.org taxonomy, where nodes are concepts and edges show superclass/subclass relationships. An enterprise can use the information in a taxonomy to find a conceptual design whose associated costs do not exceed its budget and deliver reasonably effective answers for queries. For example, assume that because an enterprise has to develop in-house annotators for concepts politician and artist, the total cost of annotating concepts in conceptual design $D_1 = \{\text{politician, artist, legislature, school, state, city}\}$ over original Wikipedia collection exceeds its budget. As some free and reasonably accurate annotators are available for concept person, e.g. nlp.stanford.edu/software/CRF-NER.shtml, the enterprise may annotate concept person using smaller amount of resources than concepts politician and artist. Hence, it may afford to annotate concepts $D_2 = \{\text{person, organization, state, city}\}$ from this collection. Thus, the enterprise may choose to annotate the data set using $D_2$ instead of $D_1$. Figure 4 demonstrates the annotated version of the excerpts of Wikipedia articles in Figure 1 using conceptual design $D_2$.

Intuitively, a query interface can disambiguate fewer queries over the data fragment in Figure 4 than the one in Figure 2. For instance, if a user asks for information about John Adams, the politician, over Figure 4, the query interface may return the document that contains information about John Adams, the artist, as an answer as both entities are annotated as person. Nonetheless, the annotated data set in Figure 4 can still help the query interface to disambiguate some queries. For example, the query interface can recognize the occurrence of entity John Adams, the school, from the people named John Adams in Figure 4. Thus, it can answer queries about the school entity over this data fragment effectively. Clearly, an enterprise would like to select a conceptual design whose required time and/or resources for extraction do not exceed its budget and most improves the effectiveness of answering queries. We call such a conceptual design for an annotated data set, a cost-effective conceptual design for the data set.

1.4 Our Contributions

Currently, concept annotation experts use their intuitions to discover cost-effective conceptual designs from taxonomies. Because most taxonomies contain hundreds of concepts [14], this approach does not scale for real-world applications. In this paper, we introduce and formalize the problem of finding cost-effective conceptual designs from taxonomies and propose algorithms to solve the problem in general and interesting special cases. To this end, we make the following contributions.

- We develop a theoretical framework that quantifies the amount of improvement in the effectiveness of answering queries by annotating a subset of concepts from a taxonomy. Our framework takes into account possibility of error in concept annotation.
- We introduce and formally define the problem of cost-effective conceptual design over tree-shaped taxonomies and show it to be NP-hard.
• We propose an efficient approximation algorithm, called the level-wise algorithm, and prove that it has a bounded worst-case approximation ratio in an interesting special case of the problem. We also propose an exact algorithm for the problem with pseudo polynomial running time.

• We further define the problem over taxonomies that are directed acyclic graphs and prove that given a generally accepted hypothesis, there is no approximation algorithm with reasonably small approximation ratio and no algorithm with pseudo polynomial running time for this problem. 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 formal framework using a large scale real-world data set, Wikipedia, real-world taxonomies [14], and a sample of a real-world query workload. Our results indicate that the formal framework accurately measures the amount of improvement in the effectiveness of answering queries using a subset of concepts from a taxonomy.

• We perform extensive empirical studies to evaluate the accuracy and efficiency of the proposed algorithms over real-world data sets, taxonomies, and query workload. Our results indicate that the pseudo polynomial algorithm is generally able to deliver more effective schemas that the level-wise algorithm in reasonable amounts of time. They further show that level-wise algorithm provides more effective conceptual designs than the pseudo polynomial algorithm if the distribution of concepts in queries is skewed.

The paper is organized as follows. Section 2 reviews the related work. Section 3 formalizes the problem of cost-effective conceptual design over a tree-shaped taxonomy and show that it is NP-hard. Section 4 describes an efficient approximation algorithm with bounded approximation ratio in an interesting special case of the problem. Section 5 proposes a pseudo-polynomial algorithm for the problem in general case. Section 6 defines the problem over taxonomies that are directed acyclic graphs and provides interesting hardness results for this setting. Section 8 concludes the paper.

2 Related Work

Researchers have noticed the overheads and costs of curating and organizing large data sets [13] [20] [19]. For example, some researchers have recently considered the problem of selecting data sources for fusion such that the marginal cost of acquiring a new data source does not exceed its marginal gain, where cost and gain are measured using the same metric, e.g., US dollars [13]. Our work extends this line of research by finding cost-effective designs over unstructured or semi-structured data sets, which help users query explore these data sets more easily. We also use a different model, where the cost and benefit of annotating concepts can be measured in different units.

There is a large body of work on building large-scale data management systems for annotating and extracting entities and relationships from unstructured and semi-structured data sources [9] [12]. In particular, researchers have proposed several techniques to optimize the running time, required computational power, and/or storage consumption 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 [18] [19] [17] [20]. Our work complements these efforts by finding a cost-effective set of concepts for annotation in the design phase. Further, our framework can handle other types of costs in creating and maintaining annotated data set other than computational overheads.

Researchers have examined the problem of selecting a cost effective subset of concepts from a set of concepts for annotation [24]. Concepts in many real-world domains, however, are maintained in taxonomies rather than unorganized sets. We build on this line of work by considering the superclass/subclass relationships between concepts in taxonomies to find cost-effective designs. Because taxonomies have richer structures than sets of concepts, they present new opportunities for finding cost-effective designs. For instance, an enterprise may not have sufficient budget to annotate a concept C in a dataset, but have adequate resources to annotate occurrences of a superclass of C, such as D, in the dataset. Hence, to answer queries about entities of C, the query interface may examine only the documents that contain mentions to the entities of D. As the query interface does not need to consider all documents in the data set, it is more likely that it returns relevant answers for queries about C. Because the algorithms proposed in [24] do not consider superclass/subclass relationships between concepts, one cannot use them to find cost-effective designs over taxonomies. Moreover, as we prove in this paper, it is more challenging and harder to find cost-effective designs over taxonomies than over sets of concepts.

Researchers have proposed methods to semi-automatically construct or expand taxonomies by discovering new concepts from large text collections [11]. We, however, focus on the problem of annotating instances of the concepts in a given taxonomy over an unstructured or semi-structured data set.

Conceptual design has been an important problem in data management from its early days [15]. Generally, conceptual designs have been created manually by experts who identify the relevant concepts in a domain of interest. Because an enterprise may not afford to annotate the instances of all relevant concepts in a domain, this approach cannot be applied to large-scale concept annotation. As a matter of fact, our empirical studies indicate that adapting this approach does not generally return cost-effective conceptual designs for annotation. Researchers have studied the problem of predicting the costs of developing or maintaining pieces of software [7]. Our work is orthogonal to the methods used for estimating the
costs of creating and maintaining concept annotation modules.

3 Cost-Effective Conceptual Design

3.1 Basic Definitions

Similar to previous works, we do not rigorously define the notion of named entity \[1\]. We define a named entity (entity for short) as a unique name in some (possibly infinite) domain. A concept is a set of entities, i.e., its instances. Some examples of concepts are person and country. An entity of concept person is Albert Einstein and an entity of concept country is Jordan. Concept \( C \) is a subclass of concept \( D \) iff we have \( C \subseteq D \). 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 3 we will explore the taxonomies that are directed acyclic graphs. Formally, we define taxonomy \( X = (R, C, \mathcal{R}) \) as a rooted tree, with root concept \( R \), vertex set \( C \) and edge set \( \mathcal{R} \). \( C \) is a finite set of concepts. For \( C, C' \in C \) we have \( (C, C') \in \mathcal{R} \) iff \( C' \) is a subclass of \( C \). Every concept in \( C \) that is not a superclass of any other concept in \( C \) is a leaf concept. The leaf concepts are leaf nodes in taxonomy \( X \). For instance, concepts athlete and artist are leaf concepts in Figure 5. Let \( ch(C) \) denote the children of concept \( C \). For the sake of simplicity, we assume that \( \bigcup_{C \in \mathcal{R}} \{D \in C | D \text{ is a leaf concept}\} \) for all \( C \) in a taxonomy.

Each data set is a set of documents. Data set \( DS \) is in the domain of taxonomy \( X \) iff some entities of concepts in \( X \) appear in some documents in \( DS \). For instance, the set of documents in Figure 1 are in the domain of the taxonomy shown in Figure 5. An entity in \( X \) may appear in several documents in a data set. For brevity, we refer to the occurrences of entities of a concept in a data set as the occurrences of the concept in the data set.

A query \( q \) over \( DS \) is a pair \( (C, T) \), where \( C \in C \) and \( T \) is a set of terms. Some example queries are \( (\text{person, Michael Jordan}) \) or \( (\text{location, Jordan}) \). This type of queries has been widely used to search and explore annotated data sets [8, 10, 24]. Empirical studies on real world query logs indicate that the majority of entity centric queries refer to a single entity [25]. In this paper, we consider queries that refer to a single entity. Considering more complex queries that seek information about relationships between several entities requires more sophisticated models and algorithms and more space than a paper. It is also an interesting topic for future work.

3.2 Conceptual Design

Conceptual design \( S \) over taxonomy \( X = (R, C, \mathcal{R}) \) is a non-empty subset of \( C - \{R\} \). For brevity, in the rest of the paper, we refer to conceptual design as design. A design divides the set of leaf nodes in \( C \) into some partitions, which are defined as follows.

**Definition 3.1.** Let \( S \) be a design over taxonomy \( X = (R, C, \mathcal{R}) \), and let \( C \in S \). We define the partition of \( C \) as a subset of leaf nodes of \( C \) with the following property. A leaf node \( D \) is in the partition of \( C \) iff \( D \subseteq C \) or \( C \) is the lowest ancestor of \( D \) in \( S \).

Let function part map each concept into its partition.

**Example 3.2.** Consider the taxonomy described in Figure 5. Let design \( S \) be \( \{\text{agent, person}\} \). The partitions of \( S \) are \( \{\text{artist, politician, athlete}\} \) and \( \{\text{school, legislature}\} \). Also, part(person) = \( \{\text{artist, politician, athlete}\} \) and part(agent) = \( \{\text{school, legislature}\} \).

For each design \( S \), the set of leaf concepts that do not belong to any partition are called free concepts and denoted as free(\( S \)). These concepts neither belong to \( S \) nor are descendant of a concept in \( S \).

Figure 5: The concepts in red, agent and person, 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 3.3. Again consider design \{person, agent\} over the taxonomy described in Figure 5. The free concepts of \( S \) are \{state, city\} as they are not in any partition of \( S \).

Let \( DS \) be a data set in the domain of taxonomy \( \mathcal{X} = (R, C, R) \) and \( S \) be a design over \( \mathcal{X} \). \( S \) is the design of data set \( DS \) iff for all concept \( C \in S \), all occurrences of concepts in the partition of \( C \) are annotated by \( C \). In this case, we say \( DS \) is an instance of \( S \). For example, consider the design \( T = \{person, organization\} \) over the taxonomy in Figure 3. The data set in Figure 4 is an instance of \( T \) as all instances of concepts athlete, artist and politician, that belong to the partition of person, are annotated by person and all instances of concepts school and legislature, that constitute the partition of organization, are annotated by organization in the data set.

3.3 Design Queriability

Let \( Q \) be a set of queries over data set \( DS \). Given design \( S \) over taxonomy \( \mathcal{X} = (R, C, R) \), we would like to measure the degree by which \( S \) improves the effectiveness of answering queries in \( Q \) 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 \( Q \) more effectively. As most entity-centric information needs are precision-oriented [5][10], we use the standard metric of precision at \( k \) (\( p@k \) for short) to measure the effectiveness of answering queries over structured data sets [22]. The value of \( p@k \) is the fraction of relevant answers in the top \( k \) returned answers for the query. We average the values of \( p@k \) over queries in \( Q \) to measure the amount of effectiveness in answering queries in \( Q \). The problem of design in order to maximize other objective functions, such as recall, is an interesting subject for future work.

Let \( Q : (C, T) \) be a query in \( Q \) such that \( C \) belongs to the partition of \( P \in S \). The query interface may consider only the documents that contain information about entities annotated by \( P \) to answer \( Q \). For instance, consider query \( Q_1 = (politician, JohnAdams) \) over data set fragment in Figure 4 whose design is \( \{person, organization\} \). The query interface may examine only the entities annotated by \( person \) in this data set 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 \( Q \). It may further rank them according to its ranking function, such as the traditional TF-IDF scoring methods [22]. Our model is orthogonal to the method used to rank the candidate answers for the query.

The query interface still has to examine all documents that contain some mentions to the entities annotated by concept \( P \) to answer \( Q : (C, T) \). Nevertheless, only a fraction of these documents may contain information about entities of \( C \). For instance, to answer query \( (politician, JohnAdams) \) over the data set fragment in Figure 4 the query interface has to examine all documents that contain instances of concept person. Some documents in this set have matching entities form concepts other than politician, such as John Adams, the artist. We like to estimate the fraction of the results for \( Q : (C, T) \) that contains a matching entity in 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, and therefore, a larger value of \( p@k \) for \( Q \).

Let \( d_{DS}(C) \) denote the fraction of documents that contain entities of concept \( C \) in data set \( DS \). We call \( d_{DS}(C) \) the frequency of \( C \) over \( DS \). When \( DS \) is clear from the context, we denote the frequency of \( C \) as \( d(C) \). We want to compute the fraction of the returned answers for query \( Q : (C, T) \) that contain a matching instance of concept \( C \). These entities are annotated by concept \( P \), such that \( C \) is in the partition of \( P \). Let \( d(P) \) be the total frequency of leaf concepts in the partition of \( P \). The fraction of these documents that contain information about \( C \) is \( \frac{d(C)}{d(P)} \). The larger this fraction is, the more likely it is that query interface returns more documents about entities of concept \( C \) for query \( Q : (C, T) \). Thus, it is more likely for query interface to return relevant answers for \( Q \) and improve its \( p@k \). For instance, assume that the mentions to the entities of concept artist appear more frequently in data set \( DS \) than the ones of concept politician. Also assume that we only annotate person from \( DS \). Given query \( (politician, JohnAdams) \) it is more likely for articles about John Adams, the artist, to appear in the top-ranked answers than about John Adams, the politician.

We call the fraction of queries in \( Q \) whose concept is \( C \) the popularity of \( C \) in \( Q \). Let \( u_Q \) be the function that maps concept \( C \) to its popularity in \( Q \). When \( Q \) is clear from the context, we simply use \( u \) instead of \( u_Q \). The degree of improvement in value of \( p@k \) in answering queries of concept \( C \) over \( DS \) is proportional to \( \frac{u(C) \cdot d(C)}{d(P)} \). Hence, the amount of the contribution of queries of the concepts in partition of \( P \) to the value of \( p@k \) will be:

\[
\sum_{C \in \text{part}(P)} \frac{u(C) \cdot d(C)}{d(P)}
\]

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 \( p@k \) value over queries in \( Q \).

Annotators, however, may make mistakes in identifying the correct concepts of entities in a collection [10]. An annotator may recognize some appearances of entities from concepts that are not \( P \) as the occurrences of entities in \( P \). For instance, 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 \( \text{pr}_{DS}(P) \). When \( DS \) is clear from the context, we show
Definition 3.4. The queriability of design $S$ from taxonomy $X$ over data set $DS$ is

$$QU(S) = \sum_{P \in S} \sum_{C \in \text{part}(P)} \frac{u(C)d(C)}{d(P)} \cdot pr(P).$$

(1)

Next, we compute the amount of improvement that $S$ provides for queries whose concepts do not belong to any partition, i.e., free concepts. If concept $C$ is a free concept with regard to design $S$, the query interface has to examine all documents in the collection to answer $Q : (C, T)$. Thus, if $C$ is a free concept, the fraction of returned answers for $Q$ that contains a matching instance of concepts $C$ is $d(C)$. Using equation $1$, we formally define the function that estimates the likelihood of improvement for the value of $p \oplus k$ for all queries in a query workload over a data set annotated by design $S$.

**Theorem 3.6.** Unfortunately, the CECD problem cannot be solved in polynomial time in terms of input size unless $P = NP$.

Proof. The problem of CECD can be reduced to the problem of choosing cost-effective concepts from a set of concepts by creating a taxonomy $X = (R, C, \mathcal{R})$ where all nodes except for $R$ are leaf concepts, i.e. leaves. Since the problem of choosing cost-effective concepts from a set of concepts is NP-hard [27], CECD will be NP-hard.

Because CECD is NP-hard, we propose and study efficient approximation and pseudo-polynomial algorithms to solve it.

4 Level-Wise Algorithm

Level-wise algorithm solves 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 algorithm proposed in [27], called approximate popularity maximization (APM for short), for finding the cost-effective
subset of concepts over a set of concepts. It eventually delivers the design with largest queriability across all levels in the taxonomy.

Precisely, let \(C[i]\) be the set of all concepts of depth \(i\) in \(X = (R, C, R)\). For any concept \(C \in C[i]\), we define its popularity \(u(C)\) to be the total popularity of its descendant leaf concepts in \(X\). Level-wise algorithm calls the APM algorithm to find the cost-effective subset of concepts for every \(C[i]\). It also computes the queriability of the design that contains only the most popular leaf concept, i.e., the leaf concept with maximum \(u\) value. It then compares various selected designs across \(C[i]\)'s and returns the answer with maximum queriability as its solution for the problem of CECD over taxonomy \(X\). Figure 6 illustrates the level-wise algorithm. Let \(|C|\) denote the number of concepts in taxonomy \(X\). The APM algorithm runs in \(O(|C| \log |C|)\). Thus, the time complexity of level-wise algorithm is \(O(h|C| \log |C|)\) over taxonomy \(X\).

In addition to being efficient, level-wise algorithm also has bounded and reasonably small worst-case approximation ratio for an interesting case of CECD problem. 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. Our empirical results in Section 7 shows that this strategy returns effective designs in the cases that the budget is relatively small. In this case, we should restrict the feasible solutions in the CECD problem to be disjoint. 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 [30]. We show that the level-wise algorithm has a bounded and reasonably small worst-case approximation ratio for CECD with disjoint design given that distribution of concept frequencies follows a power law distribution. The following lemma bounds the queriability that is obtained from the free concepts in any solution given that distribution of concept frequencies follows a power law distribution.

**Lemma 4.1.** Let \(C_{\text{max}}\) be the leaf concept in taxonomy \(X = (R, C, R)\) with maximum \(u\) value and let assume that distribution of \(u\) over leaf concepts follows a power law distribution. Let \(S\) be any schema. Then,

\[
QU(\text{free}(S)) \leq 2u(C_{\text{max}}) \log |C|.
\]

**Proof.** We have:

\[
\sum_{C \in \text{free}(S)} u(C) d(C) \leq u(C_{\text{max}}) \sum_{C \in \text{free}(S)} d(C).
\]

Since the frequencies of leaf concepts in \(X\) follow a “power law” distribution,

\[
\sum_{C \in \text{leaf}(C)} d(C) \leq 1 + \log(|\text{leaf}(C)|),
\]

where \(\text{leaf}(C)\) is the set leaf concepts in \(C\) and \(|\text{leaf}(C)|\) is the number of such concepts. Since \(|\text{leaf}(C)| \leq |C|\),

\[
QU(\text{free}(S)) \leq \sum_{C \in \text{free}(S)} u(C) d(C) \leq (1 + \log |C|) u(C_{\text{max}}) \leq 2u(C_{\text{max}}) \log |C|.
\]

\(\Box\)

**Theorem 4.2.** Let \(X = (R, C, R)\) be a taxonomy with height \(h\) and the minimum accuracy of \(pr_{\text{min}} = \min_{C \in C} pr(C)\). The Level-wise algorithm is a \(O(\frac{h + \log |C|}{\text{pr}_{\text{min}}})\)-approximation for the CECD problem with disjoint solution on \(X\) and budget \(B\) given that the distribution of frequencies in \(C\) follows a power law distribution.

**Proof.** Let \(S^*\) be a disjoint schema over \(X\) with total cost at most \(B\) that maximizes \(QU\) function. Let \(S^*[i]\) be the set of concepts in \(S^*\) of depth \(i\). By the definition of disjointness, \(\text{part}(S^*[i]) \cap \text{part}(S^*[j]) = \emptyset\), for all \(1 \leq i, j \leq h\). It follows:

\[
QU(S^*) = \sum_{1 \leq i \leq h} QU(S^*[i]) + QU(\text{free}(S^*)),
\]

where \(QU(\text{free}(S^*)) = \sum_{C \in \text{free}(S^*)} u(C) d(C)\) is the queriability obtained from the free concepts in \(S^*\).

We consider two possible cases. First, assume that \(\sum_{i=1}^{h} QU(S^*[i]) \geq QU(\text{free}(S^*))\). It immediately follows that the level-wise algorithm output gives a \(2h/\text{pr}_{\text{min}}\)-approximation. In the other case in which \(QU(\text{free}(S^*)) \geq \sum_{1 \leq i \leq h} QU(S^*[i])\), by Lemma 4.1 extracting the concept with the maximum \(u\) value gives a \(4 \log (|C|)/\text{pr}_{\text{min}}\)-approximation. These two cases together imply that we have an \(O(\frac{h + \log |C|}{\text{pr}_{\text{min}}})\)-approximation. \(\Box\)

The value of \(\text{pr}_{\text{min}}\) is generally large because concept annotation algorithm are reasonably accurate [9, 23].
Formally given budget \( X \) leaf concepts in the subproblems defined over the subtrees rooted at the children of \( C \). Queriability of the partitions obtained by \( X \) programming technique. The main idea is to define the CECD problem over all subtrees of the given taxonomy \( \mathcal{X} = (R, C, \mathcal{R}) \). Next we show that in order to solve the subproblem defined over the subtree rooted at \( C \), it is enough to solve the subproblems defined over the subtrees rooted at the children of \( C \).

Let \( \text{child}(C) \) be the set of all children of the concept \( C \) in \( \mathcal{X} \). Moreover, let \( \mathcal{X}_C \) be the subtree of \( \mathcal{X} \) rooted at \( C \). Formally given budget \( B_C \), the subproblem over \( \mathcal{X}_C \) is to find a design \( S_C \subseteq \mathcal{X}_C \) whose total cost is at most \( B_C \) and the queriability of the partitions obtained by \( S_C \) is the maximum. Note that by annotating \( S_C \) in \( \mathcal{X}_C \) there may exist a set of leaf concepts in \( \mathcal{X}_C \) that do not belong to any of \( \text{part}(S) \) for \( S \in \mathcal{S} \). Let \( \text{nullPart}(S_C, C) \) denotes the leaf concepts of \( \mathcal{X}_C \) that are not assigned to any partition of \( S_C \).

In order to compute the maximum queriability of the best design in \( \mathcal{X}_C \), one of the cases we should consider is the one in which \( C \) is annotated. To apply dynamic programming in this case we need to evaluate the queriability of \( \text{part}(C) \) which is \( \sum_{Ch \in \text{child}(C)} \sum_{C' \in \text{nullPart}(S_{Ch}, Ch)} u(C')d(C') \). Thus besides the total queriability of partitions in \( \mathcal{X}_C \), we should compute the value of \( \sum_{C' \in \text{nullPart}(S_{Ch}, Ch)} u(C')d(C') \). All together we are required to solve the subproblem \( Q \) defined over the subtree rooted at \( C \) with parameter \( B_C \) and \( N_C \) where \( B_C \) denotes the available budget for annotating concepts in \( \mathcal{X}_C \) and \( N_C \) denotes the value of \( \sum_{C' \in \text{nullPart}(S_{Ch}, C)} u(C')d(C') \).

Further we assume that \( u(C), d(C), \) and \( w(C) \) are positive integers for each \( C \in \mathcal{C} \). In Section 7 we show that the algorithm can handle real values with scaling techniques in expense of reporting a near optimal solution instead of an optimal one. We define \( D = \sum_{C \in \text{leaf}(C)} d(C), U = \sum_{C \in \text{leaf}(C)} u(C) \). Let \( B_{\text{total}} \) denote the total available budget. We propose an algorithm whose time complexity is polynomial in \( U, D, B_{\text{total}}, \) and \( |C| \).

We have the following recursive rules for the non-leaf concepts in \( \mathcal{C} \) based on the value of \( Q \) for their children.

### Level-wise

〈Input: \( \langle X \rangle \)〉

\[
\text{sol}_{\text{level1}} \leftarrow 0 \quad \text{and} \quad \text{sol}_{\text{max}} \leftarrow 0
\]

〈Return the output of the best level〉

For \( i = 0 \) to \( h \) do

- For each concept \( C \) in distance \( i \) from the root
  - \( C_i \leftarrow C_i \cup C \)
  - \( \text{sol}_i \leftarrow \) approximate solution over \( (C_i) \)
  - \( \text{sol}_{\text{level1}} \leftarrow \max(\text{sol}_{\text{level1}}, \text{sol}_i) \)

〈Most Popular Leaf Concept Only〉

Let \( C_{\text{max}} \) be the leaf concept with the largest \( u \) value.

\[
\text{sol}_{\text{max}} \leftarrow u(C_{\text{max}}) + \sum_{C \in \text{free}(C_{\text{max}})} u(C)d(C)
\]

Return the best of \( \text{sol}_{\text{level1}} \) and \( \text{sol}_{\text{max}} \)

---

**Figure 6:** Level-wise algorithm.

### 5 Pseudo-polynomial Time Algorithm

In this section we describe a pseudo-polynomial time algorithm for the CECD problem over tree taxonomies. As many other optimization problems on the tree structure, one approach is to find an optimal solution bottom-up using dynamic programming technique. The main idea is to define the CECD problem over all subtrees of the given taxonomy \( \mathcal{X} = (R, C, \mathcal{R}) \). Next we show that in order to solve the subproblem defined over the subtree rooted at \( C \), it is enough to solve the subproblems defined over the subtrees of the given taxonomy \( \mathcal{X} = (R, C, \mathcal{R}) \). Next we show that in order to solve the subproblem defined over the subtree rooted at \( C \), it is enough to solve the subproblems defined over the subtrees rooted at the children of \( C \).

Let \( \text{child}(C) \) be the set of all children of the concept \( C \) in \( \mathcal{X} \). Moreover, let \( \mathcal{X}_C \) be the subtree of \( \mathcal{X} \) rooted at \( C \). Formally given budget \( B_C \), the subproblem over \( \mathcal{X}_C \) is to find a design \( S_C \subseteq \mathcal{X}_C \) whose total cost is at most \( B_C \) and the queriability of the partitions obtained by \( S_C \) is the maximum. Note that by annotating \( S_C \) in \( \mathcal{X}_C \) there may exist a set of leaf concepts in \( \mathcal{X}_C \) that do not belong to any of \( \text{part}(S) \) for \( S \in \mathcal{S} \). Let \( \text{nullPart}(S_C, C) \) denotes the leaf concepts of \( \mathcal{X}_C \) that are not assigned to any partition of \( S_C \).

In order to compute the maximum queriability of the best design in \( \mathcal{X}_C \), one of the cases we should consider is the one in which \( C \) is annotated. To apply dynamic programming in this case we need to evaluate the queriability of \( \text{part}(C) \) which is \( \sum_{Ch \in \text{child}(C)} \sum_{C' \in \text{nullPart}(S_{Ch}, Ch)} u(C')d(C') \). Thus besides the total queriability of partitions in \( \mathcal{X}_C \), we should compute the value of \( \sum_{C' \in \text{nullPart}(S_{Ch}, Ch)} u(C')d(C') \). All together we are required to solve the subproblem \( Q \) defined over the subtree rooted at \( C \) with parameter \( B_C \) and \( N_C \) where \( B_C \) denotes the available budget for annotating concepts in \( \mathcal{X}_C \) and \( N_C \) denotes the value of \( \sum_{C' \in \text{nullPart}(S_{Ch}, C)} u(C')d(C') \).

Further we assume that \( u(C), d(C), \) and \( w(C) \) are positive integers for each \( C \in \mathcal{C} \). In Section 7 we show that the algorithm can handle real values with scaling techniques in expense of reporting a near optimal solution instead of an optimal one. We define \( D = \sum_{C \in \text{leaf}(C)} d(C), U = \sum_{C \in \text{leaf}(C)} u(C) \). Let \( B_{\text{total}} \) denote the total available budget. We propose an algorithm whose time complexity is polynomial in \( U, D, B_{\text{total}}, \) and \( |C| \).

We have the following recursive rules for the non-leaf concepts in \( \mathcal{C} \) based on the value of \( Q \) for their children.

![Figure 7: The concepts in red denote the ones that are picked in the design. (a), (b) and (c) show three different types of the subproblems required to solve in order to compute \( Q[C, B, 0] \).](image-url)
\[ Q[C,B,0] = \max_{B,N} \left\{ \max_{Ch \in \text{child}(C)} \sum_{Ch,B(Ch),N(Ch)} Q[Ch,B(Ch),N(Ch)] + \frac{pr(C)}{d(C)} \sum_{Ch \in \text{child}(C)} N(Ch) \right\} \]

For each \( Ch, B(Ch), B'(Ch) \) and \( N(Ch) \) are integer values satisfying the following conditions: (1) \( B = w(C) + \sum_{Ch \in \text{child}(C)} B(Ch) \), (2) \( B = \sum_{Ch \in \text{child}(C)} B'(Ch) \), and (3) \( UD \geq \sum_{Ch \in \text{child}(C)} N(Ch) \).

The first term in the recursive rule corresponds to the case in which we select concept \( C \) in the output design ((b) and (c) in Figure 7) and the second term corresponds to the case in which for any child of \( C, \text{nullPart}(Ch) = \emptyset \) (a) in Figure 7). In a design \( X_C \) in \( X \) with the maximum queriability and empty \( \text{nullPart} \) whose total cost is \( B \), either \( C \) is selected in the design and the budget \( B - w(C) \) is divided among the children of \( C \) (first term of the above rule), or the whole budget \( B \) is divided among the children of \( C \) and all leaf concepts of \( X_C \) is assigned to a proper descendant of \( C \) in the design (second term of the above rule).

Similarly, for the case in which \( N \neq 0 \) we have the following recursive rule:

\[ Q[C,B,N] = \max_{B,N} \sum_{Ch \in \text{child}(C)} Q[Ch,B(Ch),N(Ch)] \]

where \( B = \{B(Ch)|Ch \in \text{child}(C)\} \) and \( N = \{N(Ch)|Ch \in \text{child}(C)\} \) such that \( B = \sum_{Ch \in \text{child}(C)} B(Ch) \) and \( N = \sum_{Ch \in \text{child}(C)} N(Ch) \). For each leaf concept \( C_t \) in \( C \), we have the following:

- \( Q[C_t,B,N] = 0 \) if \( N = u(C_t)d(C_t) \) and \( -\infty \) otherwise.
- \( Q[C_t,B,0] = pr(C_t)u(C_t) \) if \( B \geq w(C_t) \) and \( -\infty \) otherwise.

The maximum value of the queriability on \( X = (R,C,R) \) is

\[ \max_{N} Q[R,B_{\text{total}},N] + N, \tag{3} \]

where \( B_{\text{total}} \) is the total available budget. The first term, \( Q[R,B_{\text{total}},N] \), denotes the profit obtained form the partitions of an optimal design and the second term corresponds to the profit obtained from the free concepts with respect to the output design.

To compute the running time of the algorithm we need to give an upper bound on the number of cells in \( Q \) and the time required to compute the value of each cell. The time to compute a single cell in \( Q \) is exponential in terms of the maximum degree of the taxonomy. Consequently, the algorithm runs much faster if the maximum degree in \( X \) is bounded by a small constant. As we show next, we can modify the taxonomy \( X \) to obtain taxonomy \( X' \) such that each concept \( C \) in \( X' \) has at most two children and the number of nodes in \( X' \) is at most twice the number of nodes in \( X \). Since each node in \( X' \) has two children, the required amount of time to compute a single cell in \( Q \) is \( O(B_{\text{total}}UD) \); at most \( B_{\text{total}} \) ways to divide the budget between the two children and at most \( UD \) ways to divide \( N \) between the two children. Since the first argument in \( Q \) can be any of the concepts in \( C \), \( N \leq UD \) and \( B \leq B_{\text{total}} \), there are \( O(B_{\text{total}}UD) \) cells to evaluate in order to compute the design with maximum queriability. Thus the total time for computing all cells in \( Q \) is \( O(|C|(B_{\text{total}}UD)^2) \).

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

![Figure 8: Transforming an input taxonomy \( X \) into a binary taxonomy. Blue square nodes correspond to dummy nodes.](image-url)
Since this transformation does not change the subset of leaf concepts in the subtree rooted in any internal node, any internal node in $X$ corresponds to a solution in $X'$ with the same cost and queriability. Since dummy nodes are too expensive to be chosen, they do not introduce any new solution to the set of feasible solutions.

**Theorem 5.1.** There is an algorithm to solve the CECD problem over taxonomy $X = (R, C, \mathcal{R})$ with budget $B$ in $O(|C|B^2U^2D^2)$.

Table 1 presents a summary of proposed algorithms for the CECD problem.

| Algorithm               | Approximation ratio | Running time                  |
|------------------------|---------------------|--------------------------------|
| Level-wise             | $O((h + \log |C|)/pr_{min})$ (Disjoint CECD) | $O(h|C|\log(|C|))$         |
| Dynamic Programming    | Pseudo-polynomial   | $O(|C|B^2U^2D^2)$            |

Table 1: Algorithms for the CECD problem.

## 6 Cost-Effective Design for DAG Taxonomies

### 6.1 Directed Acyclic Graph Taxonomies

While taxonomies are traditionally in form of trees, many of them have evolved into directed acyclic graphs (DAGs) to model more involved subclass/superclass relationships between concepts in their domains. 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 $X = (R, C, \mathcal{R})$, (DAG taxonomy for short), is a DAG, with vertex set $C$, edge set $\mathcal{R}$, and root $R$. $C$ is a set of concepts, $(D, C) \in \mathcal{R}$ iff $D, C \in C$ and $D$ is a superclass of $C$. Finally, $R$ is a node in $X$ without any superclass. A concept $C \in C$ is a leaf concept iff it has no subclass in $X$; i.e., there is not any node $D \in C$ where $(C, D) \in \mathcal{R}$. The definitions of child, ancestor, and descendant over tree taxonomies naturally extends to DAG taxonomies.

![Figure 9: Fragments of schema.org taxonomy](image)

### 6.2 Design Queriability

Design $S$ over DAG taxonomy $X = (R, C, \mathcal{R})$ is a non-empty subset of $C - \{R\}$. Due to 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 data set $DS$ be in the domain of the DAG taxonomy in Figure 9 and $S_1 = \{\text{place, organization}\}$ be a design. The query interface will examine the documents that are organized under `organization` in $DS$ to answer queries about concept `airline`. As query interface does not have sufficient information to pinpoint the entities of concept `airline` in $DS$, it may return some non-relevant answers for these queries, e.g., matching entities that are NGOs. On the other hand, because concept `hospital` is a subclass of both `place` and `organization`, its entities in $DS$ are annotated by both concepts `place` and `organization`. By examining the entities that are annotated by both `place` and `organization`, the query interface is able to identify the instances of `hospital` in $DS$. Thus, it will not return entities that belong to other concepts when answering queries about instances of `hospital`. Generally, the query interface may pinpoint instances of some concepts in the data set by considering the intersections of multiple concepts in a design over a DAG taxonomy. Hence, subsets of a design may create partitions in a DAG taxonomy. Next, we extend the notion of partitions for designs over DAG taxonomies.

**Definition 6.1.** Let $S$ be a design over DAG taxonomy $X = (R, C, \mathcal{R})$, and let $C \in C$ be a leaf concept. An ancestor $A$ of $C$ in $S$ is $C$’s direct ancestor iff one of the following properties hold.

- $A = C$.
- For each $D \in S$, 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. For instance, the set \{place, organization\} is the full-ancestor-set of the concept hospital in design \( S_1 = \{place, organization\} \), and the set \{place, local business\} is the full-ancestor-set of the concept hospital in design \( S_2 = \{place, organization, local business\} \) over the taxonomy in Figure 9.

Definition 6.2. Given design \( S \) over DAG taxonomy \( \mathcal{X} = (R, C, \mathcal{R}) \), the partition of a set of concepts \( D \subseteq \mathcal{S} \) is a set of leaf concepts \( \mathcal{L} \subseteq \mathcal{C} \) such that for every leaf concept \( L \in \mathcal{L}, D \) is the full-ancestor-set of \( L \).

For instance, hospital belongs to the partition of \{place, organization\} in \( S_1 \). But, it does not belong to the partition of \{place\}, since \{place\} is not the full-ancestor-set of hospital. The definitions of functions part and free over DAG taxonomies extend from their definitions over tree taxonomies.

Similar to tree 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 3.3, we define the queryability of conceptual design \( S \) over DAG taxonomy \( \mathcal{X} = (R, C, \mathcal{R}) \) as follows.

\[
QU(S) = \sum_{P \in \text{all-part}(S)} \frac{\sum_{C \in P} u(C)d(C)}{d(P)} + \sum_{C \in \text{free}(S)} u(C)d(C).
\]

The function \( 1 + \text{parts}(S) \subseteq 2^C \) returns the collection of all full-ancestor-sets of \( S \) in \( \mathcal{X} \). We remark that the size of \( 1 + \text{parts}(S) \) is linear, since we have at most one new partition per any leaf concept in \( \mathcal{X} \).

6.3 Hardness of Cost-Effective Design Over DAG Taxonomies

We define the CECD problem over DAG taxonomies similar to the CECD problem over tree taxonomies. Following from the \( \text{NP} \)-hardness results for CECD problem over tree taxonomy, CECD problem over DAG taxonomies is \( \text{NP}\)-hard. In this section, 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 design is disjoint.

We show that the CECD problem over a DAG taxonomy generalizes a hard problem in the approximation algorithms literature: Densest-\( k \)-Subgraph [21]. Given a graph \( G = (V, E) \), in the the Densest-\( k \)-Subgraph problem, the goal is to compute a subset \( U \subseteq V \) of size \( k \) that maximizes the number of edges in the induced subgraph of \( U \). It is known that, unless \( \text{P} = \text{NP} \), no polynomial time approximation scheme, i.e., PTAS, exists to compute the densest subgraph [21]. Moreover, there are strong evidences that Densest-\( k \)-Subgraph does not admit any approximation guarantee better than polylogarithmic factor [6, 2]. The following theorem shows that approximating the \( k \)-densest subgraph reduces to approximating CECD.

Lemma 6.3. Let \( S \) be a design over taxonomy \( \mathcal{X} = (R, C, \mathcal{R}) \) that is constructed from input \( G = (V, E) \) as above. Let \( S_v \in C \setminus S \) be a non-leaf concept. Then \( QU(S \cup \{S_v\}) \geq QU(S) \).

Proof. After annotating a non-leaf concept \( S_v \), each leaf concept \( C \) will be contained by a partition of either smaller or the same size. Since the contribution of a leaf concept \( C \) to \( QU \) only depends on the size of the partition contains \( C \) and this dependence is a non-decreasing function in terms of the size of partition, after annotating \( S_v \) the contribution of \( C \) to \( QU \) either increases or remains unchanged. Thus \( QU(S \cup \{S_v\}) \geq QU(S) \).

Theorem 6.4. A \((\log m)\)-approximation algorithm for the CECD problem over DAG taxonomy with \( m \) number of concepts implies that there is an algorithm for the Densest-\( k \)-Subgraph problem on \( G = (V, E) \) with \( n \) vertices that returns a \( O(\log n) \)-approximate solution.
Proof. Given $G$ and $k$, 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 an 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 the super class of $a_v$ and all the concepts corresponding to the incident edges to $v$ in $G$. Further, we set the budget $B$ to $k$, the cost of each non-leaf concept to 1, and the cost of each leaf concept to $k + 1$.

Note that if we select $S_v$ and $S_w$ in the design and $(u, v) \in E$, then $a_e$ will be a singleton partition. We also 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 $G$ (or equivalently the number of leaf concepts in $C$) and $n$ be the number of vertices in $G$ (or equivalently the number of non-leaf concepts in $C$). For each partition $p \in \text{part}(S)$ we set $d(p) = 1/(m \log n)$ if $|p| = 1$ and $d(p) = 1$ otherwise.

By Lemma 6.5, annotating a non-leaf concept will not decrease the queriability of the design. Since the leaf concepts are not affordable, and annotating a non-leaf concept will not decrease the total queriability, there exists an optimal design that annotates exactly $k$ non-leaf concepts. Note that in any design $\mathcal{S}$ of size $k$, the contribution of any leaf concept in a non-singleton partition (partition of size greater than one) is exactly $u \cdot d$. In what follows we show that a log $n$-approximation algorithm for the CECD problem implies a $O(\log n)$-approximation for the Densest-$k$-Subgraph problem. To this end, by contradiction, let $\mathcal{A}$ be a log $n$-approximation algorithm of CECD problem.

Let $H_G$ be the set of vertices in $G$ of whose corresponding non-leaf concepts in $\mathcal{C}$ are annotated in design $\mathcal{S}$. $E(H_G)$ denotes the set of edges with both endpoint in $H$ which corresponds to the set of edge-concepts of $\mathcal{C}$ whose both non-leaf concepts corresponding to their endpoints are annotated by $\mathcal{S}$.

Let $\mathcal{S}_\text{OPT}$ be an optimal solution of the CECD problem. Suppose that $QU(\mathcal{S}_\text{OPT}) = (t + r) \cdot m \log n + (m - t + n - r)$ where $t$ denotes the number of edges in $H_{\text{Sort}}$ and $r$ denotes the number of vertices in $H_{\text{Sort}}$ whose all incident edges are in $E(H_{\text{Sort}})$. It is straightforward to see that the corresponding leaf concepts to edges in $E(H_{\text{Sort}})$ and vertices with all incident edges in $E(H_{\text{Sort}})$ are the only singleton partitions with respect to design $\mathcal{S}_\text{OPT}$.

Now, let $\mathcal{S}_a$ be the design returned by $\mathcal{A}$ and similarly assume that $QU(\mathcal{S}_a) = (t' + r') \cdot m \log n + (m - t' + n - r')$. Since $\mathcal{A}$ is a log $n$-approximation algorithm of the CECD problem, $(t + r) \cdot m \log n + (m - t + n - r)$ is at most $\log n \cdot (t' + r') \cdot m \log n + (m - t' + n - r')$. Thus,

\[ t(m \log n - 1) \leq t'(m \log^2 n - 1) + r'm \log^2 n + (m + n) \log n. \]

Note that since the size of a feasible design is $k$, $r' \leq k$. Thus with some simplifications,

\[ t \frac{m \log n}{2} \leq t' (m \log^2 n) + km \log^2 n + 2m \log n, \]

which implies that

\[ t \leq 2t' \log n + 2k \log n + 4 \leq 5 \log n \cdot \max\{k, t'\}. \] (5)

Now consider the greedy approach of Densest-$k$-Subgraph problem such that in each step the algorithm picks a vertex $v$ and add it to the already selected set of vertices $S$ if $v$ has the maximum number of edges incident to $S$. It is easy to see that the greedy approach guarantee $k/2$ number of edges. Note that if the input graph has less than $k/2$ edges, we can solve Densest-$k$-Subgraph problem optimally by picking all edges. Using the simple greedy approach and the result returned by $\mathcal{A}$, we can find a set of $k$ vertices whose induced subgraph has at least $\max\{k/2, t\}$ number of edges. Thus by [5], we can find a $O(\log n)$-approximate solution of the Densest-$k$-Subgraph problem which completes the proof. \qed

Since the concepts in the instance of the CECD problem discussed in the proof of Theorem 6.4 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 6.4 is true even for the special cases of CECD problem over DAG taxonomies where the concepts are equally costly and/or the problem has disjoint solutions.

Figure 11 illustrates a simple example for which the level-wise algorithm is arbitrarily worse than the optimal solution over DAG taxonomies. For the sake of simplicity, let $d$ and $u$ values be positive integers. Let $u(C_1) = 4, d(C_1) = 1, u(C_3) = 1, d(C_3) = M$, $u(C_6) = M, d(C_6) = 1, u(C_7) = 1$ and $d(C_7) = M$. Also, let $w(C_1) = w(C_2) = w(C_3) = 1$, and $B = 2$. The greedy algorithm first picks $C_1$ because of its high immediate queriability, and then $C_2$ or $C_3$ (but not both of them). So its total queriability is 5. On the other hand, by picking $C_2$ and $C_3$ one may acquire $C_6$ for free, whose queriability is $M$. Since we can choose $M$ to be any number, the optimal solution can be arbitrarily better that the solution delivered by the greedy approach. Intuitively, the situation can be exacerbated to a large extent if the subset with large queriability can be obtained by intersecting more than two concepts.

7 Experiments

7.1 Experiment Setting

Taxonomies: We have selected five taxonomies of YAGO ontology version 2008-w40-2 [14] to validate our model and evaluate the effectiveness and efficiency of our proposed algorithms. YAGO organizes its concepts using superclass/
we did not select any concept from higher levels as they are very abstract. The concepts in levels lower than 7 in YAGO are too specific and rarely do they have any instance in our query workload. To validate our model, we have to compute and compare the effectiveness of answering queries using every feasible conceptual design over a taxonomy. Thus, we need taxonomies with relatively small number of concepts for our validation experiments. We have extracted three taxonomy trees with relatively small number of nodes, called $T1$, $T2$, $T3$, to use in our validation experiments. $T1$ has small number of concepts and is not balanced. $T2$ is a more balanced tree where each internal (i.e., non-leaf and non-root concepts) concept have at least two children. $T3$ is quite similar to $T2$ but is slightly deeper. We have further picked two taxonomies with larger numbers of concepts, denoted as $T4$ and $T5$, from YAGO ontology. We use all five taxonomies to evaluate the effectiveness of our proposed algorithms $T4$ and $T5$ to study the their efficiencies. Table 3 depicts the information about these taxonomies and Table 2 shows some sample concepts from each taxonomy.

**Dataset**: We have used the collection of English Wikipedia articles from the Wikipedia dump of the October 8, 2008 that is annotated by concepts from Yago ontology in our experiments [14]. This collection contains 2,666,190 articles. For each taxonomy in our sets of taxonomies, we have extracted a subset of the original Wikipedia collection where each document contains at least a mention to an entity of a concept in the taxonomy. We use each data set in the experiments over its corresponding taxonomy. Table 3 shows the properties of these five data sets. The annotation accuracies of the concepts in selected taxonomies over these data sets are between 0.8 and 0.95.

**Query Workload**: We use a subset of MSN query log whose target URLs, i.e., relevant answers, are Wikipedia articles. Each query contains between 1 to 6 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 concept associated with each query. We label each query by the concept of the matching instance in its relevant answer(s). Using this method, we create a query workload per each of our data sets. It is well known that the effectiveness of answering some queries may not improve by annotating the data set [25]. For instance, all candidate answers for a query may contain mentions to the entities of the query concept. In order to reasonably evaluate our algorithms, we have ignored the queries whose rankings remains the same over the unannotated version and the version of the data set where all concepts in the taxonomy are annotated. Table 3 shows the information about the query workloads. We use two-fold cross validation to calculate the popularities, $u$, of concepts in each taxonomy over their corresponding query workload. Because some concepts in a taxonomy may not appear in its query workload, we smooth popularities of concepts using the Bayesian m-estimate method [22]:

$$\hat{u}(C) = \frac{\hat{P}(C|QW) + m p}{m + \sum_{c} P(C|QW)}$$

where $\hat{P}(C|QW)$ is the probability that $C$ occurs in the query workload 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, $p$.

**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 [22]. Then, we apply the information about the concepts in the query and documents to return the answers whose matching instances have the same concept as the concept of the query. If the concept in the query has not been annotated from the collection, the query interface returns the list of document ranked by BM25 method without any modification. We have implemented our query interface and algorithms in Java 1.7 and performed our experiments on a Linux server with 100 GB of main memory and two quad core processors.

**Effectiveness Metric**: All queries in our query workloads have one or two relevant answers, thus, we measure the effectiveness of answering queries over a dataset using Precision at 3 ($p@3$) and mean reciprocal rank (MRR) [22]. Since our theory is more focus on precision metric, we will mainly discuss the results based on $p@3$. However, the results of both $p@3$ and MRR generally follow similar trends. We measures the statistical significance of our results using the paired-t-test at a significant level of 0.05.

**Cost Models**: We use two models for generating costs of concept annotation in our experiments. First, we assign a randomly generated cost to each concept in a taxonomy. The results reported for this model are averaged over 20 sets of random cost assignments per budget. We call this model random cost model. If there is not any reliable estimation

![Figure 11: An instance of CECD problem over DAG taxonomy](image-url)
T1: plant, animal, person, rich person, advocate
T2: document, association, club, institute, facility
T3: music, speech, literary composition, adaptation
T4: event, show, contest, group, ethnic_group
T5: person, location, language, character, accident

Table 2: Sample concepts from taxonomies T1, T2, T3, T4, and T5

| Taxonomy | #Concept | Depth | #Distinct Concepts | #Total Queries | #Documents |
|----------|----------|-------|--------------------|---------------|------------|
| T1       | 15       | 2     | 158                | 216           | 20563      |
| T2       | 17       | 3     | 98                 | 166           | 84879      |
| T3       | 66       | 4     | 1308               | 2028          | 93795      |
| T4       | 76       | 6     | 2840               | 4750          | 145061     |

Table 3: The sizes and depths of taxonomies and the sizes of their corresponding query workloads and data sets.

available for the cost of annotating concepts, an enterprise may assume that all concepts are equally costly. Hence, in our second cost model, we assume that all concepts in the input taxonomy have equal cost. We name this model uniform cost model. We use a range of budgets between 0 and 1 with a step size of 0.1 where 1 means sufficient budget to annotate all concepts in a taxonomy and 0 means no budget is available.

7.2 Validating Queriability Function

In this set of experiments, we evaluate how accurately the queriability formula measures the amount by which a design improves the effectiveness of answering queries. We use three following algorithms in these experiments.

Oracle: Given a fixed budget, Oracle enumerates all feasible designs over the input taxonomy. For each design, it computes the average $p@3$ for all queries in the query workload over the data set annotated by the design. It then picks the design with maximum value of average $p@3$. Since oracle does not use any heuristic to predict the amount of improvement in $p@3$ by a design, we use it to evaluate the accuracy of other methods that predict the amount of improvement in $p@3$ achieved by a design. We must note that due to time limitation, some results of Oracle are omitted.

Popularity Maximization (PM): Following the traditional approach toward conceptual design for databases, one may select concepts in a design that are more important for users [15]. Hence, we implement an algorithm, called PM, that given a budget enumerates all feasible designs, such as $S$, in a taxonomy and selects the one with the maximum value of

$$\sum_{p \in \text{part}(S)} \sum_{C \in p} u(C)p_r(p).$$

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 3.3. 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. As these algorithms enumerate all feasible designs, it is not possible to run them over large taxonomies. Hence, we run these algorithms over small taxonomies, namely T1, T2, and T3. Further, Oracle has to enumerate all feasible designs per each query in the query workload per each feasible design. Because each result for an algorithm using random cost model is the average of 20 different runs of the algorithm, it takes extremely long time to run oracle for this cost model. Thus, we run and report the results of oracle only for uniform cost model.

Table 5 shows the average $p@3$ achieved by Oracle, PM, and QM over taxonomies T1, T2, T3 under uniform cost model and for PM and QM under random cost model over various budgets. The values of $p@3$ shown in front of $B = 0$ is the one achieved by pure BM25 ranking without annotating any concept in the data sets.

Over all taxonomies and cost models, the designs picked by QM deliver closer $p@3$ values to the ones selected by Oracle. Particularly, in many budgets over taxonomies T1 and T3 QM delivers the same design as Oracle. The only case where the results of QM is significantly worse than the results of Oracle is for budget 0.2 over taxonomy T2. In this case, QM picks a design that consists of dramatic composition and literary composition, which are leaf concepts. However, Oracle selects writing, which is the parent of dramatic composition, literary composition, and a couple more concepts in T2. The design selected by QM is not able to improve the effectiveness of answering queries over other children of writing. This observation suggests that sometimes if the budget is relatively small, it is sometimes better to annotate rather more general concepts. With this choice, the resulting design can improve the effectiveness of answering larger number of queries. Although the amount of improvement is not much per each query, it still delivers a higher average $p@3$ over all queries. However, this result does not generally hold as QM can deliver the same designs as Oracle or designs that improve the effectiveness of answering queries close the ones selected by Oracle.
Table 4: Average $p@3$ for Oracle, PM, and QM. Statistically significant differences between PM and QM, and between Oracle and QM are marked in bold and italic, respectively.

| Taxonomy | Budget | Uniform Cost | Random Cost |
|----------|--------|--------------|-------------|
|          |        | Oracle PM QM | Oracle PM QM |
| T1       | 0.0    | 0.149 0.088 0.149 | 0.128 0.098 0.128 |
|          | 0.1    | 0.168 0.091 0.168 | 0.163 0.097 0.162 |
|          | 0.2    | 0.183 0.106 0.177 | 0.179 0.103 0.177 |
|          | 0.3    | 0.192 0.166 0.192 | 0.188 0.137 0.185 |
|          | 0.4    | 0.194 0.185 0.193 | 0.193 0.174 0.193 |
|          | 0.5    | 0.195 0.194 0.195 | 0.194 0.188 0.194 |
|          | 0.6    | 0.195 0.195 0.195 | 0.195 0.195 0.195 |
|          | 0.7    | 0.195 0.195 0.195 | 0.195 0.195 0.195 |
|          | 0.8    | 0.195 0.195 0.195 | 0.195 0.195 0.195 |
|          | 0.9    | 0.195 0.195 0.195 | 0.195 0.195 0.195 |

Table 5: Average $MRR$ for Oracle, PM, and QM. Statistically significant differences between PM and QM are marked in bold.

| Taxonomy | Budget | Uniform Cost | Random Cost |
|----------|--------|--------------|-------------|
|          |        | Oracle PM QM | Oracle PM QM |
| T1       | 0.0    | 0.362 0.197 0.362 | 0.299 0.215 0.299 |
|          | 0.1    | 0.415 0.203 0.406 | 0.401 0.218 0.398 |
|          | 0.2    | 0.459 0.227 0.459 | 0.446 0.230 0.442 |
|          | 0.3    | 0.492 0.250 0.492 | 0.478 0.252 0.470 |
|          | 0.4    | 0.306 0.299 0.304 | 0.303 0.304 0.303 |
|          | 0.5    | 0.306 0.306 0.306 | 0.306 0.306 0.306 |
|          | 0.6    | 0.306 0.306 0.306 | 0.306 0.306 0.306 |
|          | 0.7    | 0.306 0.306 0.306 | 0.306 0.306 0.306 |
|          | 0.8    | 0.306 0.306 0.306 | 0.306 0.306 0.306 |
|          | 0.9    | 0.306 0.306 0.306 | 0.306 0.306 0.306 |

QM also delivers designs that improve the $p@3$ of answering queries more than the ones picked by PM. Overall, PM annotates more general concepts from the taxonomy in order 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 form the query concept, the query interface is usually not able to improve the value of $p@3$ for a query significantly. On the other hand, QM selects the designs with less ambiguous concepts. Although its designs may not improve the ranking quality for most queries, they significantly improve the ranking quality of relatively large number of queries. For example, for budget 0.5 over taxonomy T3, PM picks a design of *written communication*, *music*, and *message*, which are relatively general concepts. QM, however, selects *statement*, which is a child of *message*, *literature*, and *dramatic composition*, which are descendants...
of written communication.

7.3 Effectiveness of Proposed Algorithms

Queriability formula needs the value of the frequency \( d \) for each concept in the input taxonomy over the data set. Nonetheless, it is not possible to exact frequencies of concepts without annotating the mentions to their entities in the data set. Similar to [27], we estimate the concept frequencies by sampling a small subset of randomly selected documents from the data set. We compute the frequency of each concept using estimation error rate of 5\% under the 95\% confidence level, which is almost 384 documents for all data sets. We also smooth the sampled frequencies using Bayesian \( m \)-estimates with smoothing parameter of 1 and uniform priors. In the remaining of the paper, we denote level-wise algorithm as \( LW \) and dynamic programming algorithm as \( DP \) for brevity.

\( LW \) and \( DP \) sometimes do not exhaust all the available budget. In these cases, we select the remaining concepts from the taxonomy in descending order of the ratio of their popularities to their costs till there is no budget left. Since \( DP \) assumes popularity, frequency, and cost to be positive integers, we use a standard scaling technique to convert the values of popularity, frequency, and cost of every concept in the input taxonomy to positive integers [28]. More precisely, let \( v_{\text{max}} \) be the maximum popularity of leaf concepts in the taxonomy and \( \varepsilon < 1 \), we scale \( u(C) \) as \( \tilde{u}(C) = \left\lfloor \frac{u(C)}{v_{\text{max}}} \right\rfloor \). We use similar techniques to scale the values of \( d(C) \) and \( w(C) \). Intuitively, the smaller the value of \( \varepsilon \) is the more exact result \( DP \) will deliver. However, it will take longer to run the algorithm for smaller values of \( \varepsilon \) as the range of \( U, D \), and \( B_{\text{total}} \) will become larger. We set the value of \( \varepsilon \) to 0.1 for the experiments in this section. We report the sensitivity of the results of \( DP \) to the choices of values for \( \varepsilon \) in Section 7.4.

Table 6 and 7 show the values of average \( p@3 \) for \( LW \) and \( DP \) over all taxonomies and cost models. We do not show the values of average \( p@3 \) for budgets greater than 0.7 for T5 as they are equal to the values reported for the budget of 0.7. Overall, the designs returned by \( DP \) improve the effectiveness of answering queries for all taxonomy except for T5 more than the designs returned by \( LW \). Because \( DP \) explores more feasible designs, it will have a better chance of finding more effective designs. \( LW \), however, returns the designs that delivers larger values of \( p@3 \) when the budget is relatively small over T4. Give a small budget, it is more reasonable to annotate disjoint concepts to improve the effectiveness of a larger number of queries. Nevertheless, if the budget is relatively large there are more choices of designs and more effective designs are not necessarily disjoint. Thus, \( DP \) finds more effective designs than \( LW \) as shown in table 7 for T4. \( LW \) also delivers designs that with larger values of average \( p@3 \) for all budgets over T5. The distribution of popularities of leaf concepts in T5 follow a very skewed distribution where the concept of more than 65\% of queries is \textit{person}. Since the distribution of concept frequencies over the data set is not very skewed, the designs that contain the most popular concepts generally deliver more effective answers to queries. Because of its greedy approach, \( LW \) finds the most popular concept(s). Since \( DP \) has to use scaling, it cannot explore all feasible designs and may miss some very popular concepts. Nevertheless, if the budget is relatively large \( DP \) is able to find designs that are as effective as the designs delivered by \( LW \) as shown in table 7 for T5.

Table 6: Average \( p@3 \) for \( LW \) and \( DP \) over T1, T2 and T3. Statistically significant difference between \( LW \) and \( DP \) are marked in bold and italic, respectively.
Table 7: Average $p_{\theta3}$ for LW and $\text{DP}_{0.1\epsilon}$ over T4 and T5. Statistically significant difference between LW and DP are marked in bold and italic, respectively.

| Taxonomy | Budget | Uniform Cost | Random Cost |
|----------|--------|--------------|-------------|
|          |        | LW | 10$^6$ | LW | 10$^6$ |
| T4       | 0.1    | 0.221 | 0.223 | 0.229 | 0.228 |
|          | 0.2    | 0.274 | 0.250 | 0.271 | 0.255 |
|          | 0.3    | 0.283 | 0.261 | 0.283 | 0.270 |
|          | 0.4    | 0.285 | 0.278 | 0.285 | 0.282 |
|          | 0.5    | 0.285 | 0.291 | 0.285 | 0.291 |
|          | 0.6    | 0.285 | 0.291 | 0.285 | 0.291 |
|          | 0.7    | 0.285 | 0.291 | 0.285 | 0.292 |
|          | 0.8    | 0.285 | 0.292 | 0.285 | 0.292 |
|          | 0.9    | 0.285 | 0.292 | 0.285 | 0.292 |
| T5       | 0.1    | 0.211 | 0.210 | 0.217 | 0.212 |
|          | 0.2    | 0.237 | 0.225 | 0.237 | 0.226 |
|          | 0.3    | 0.244 | 0.233 | 0.245 | 0.235 |
|          | 0.4    | 0.247 | 0.239 | 0.247 | 0.242 |
|          | 0.5    | 0.248 | 0.246 | 0.248 | 0.246 |
|          | 0.6    | 0.248 | 0.247 | 0.248 | 0.247 |
|          | 0.7    | 0.248 | 0.248 | 0.248 | 0.248 |
|          | 0.8    | 0.248 | 0.248 | 0.248 | 0.248 |
|          | 0.9    | 0.248 | 0.248 | 0.248 | 0.248 |

Table 8: Average $MRR$ for LW and $\text{DP}_{0.1\epsilon}$ over T1, T2 and T3. Statistically significant differences between LW and DP are marked in bold.

| Taxonomy | Budget | Uniform Cost | Random Cost |
|----------|--------|--------------|-------------|
|          |        | LW | 10$^6$ | LW | 10$^6$ |
| T1       | 0.1    | 0.203 | 0.200 | 0.195 | 0.202 |
|          | 0.2    | 0.203 | 0.221 | 0.215 | 0.284 |
|          | 0.3    | 0.203 | 0.394 | 0.209 | 0.317 |
|          | 0.4    | 0.227 | 0.438 | 0.243 | 0.424 |
|          | 0.5    | 0.440 | 0.492 | 0.340 | 0.460 |
|          | 0.6    | 0.444 | 0.501 | 0.433 | 0.497 |
|          | 0.7    | 0.497 | 0.507 | 0.473 | 0.503 |
|          | 0.8    | 0.507 | 0.507 | 0.503 | 0.507 |
|          | 0.9    | 0.507 | 0.507 | 0.507 | 0.507 |
| T2       | 0.1    | 0.504 | 0.479 | 0.506 | 0.541 |
|          | 0.2    | 0.574 | 0.616 | 0.581 | 0.616 |
|          | 0.3    | 0.586 | 0.641 | 0.607 | 0.647 |
|          | 0.4    | 0.615 | 0.670 | 0.646 | 0.683 |
|          | 0.5    | 0.685 | 0.713 | 0.709 | 0.721 |
|          | 0.6    | 0.761 | 0.753 | 0.755 | 0.749 |
|          | 0.7    | 0.762 | 0.757 | 0.763 | 0.759 |
|          | 0.8    | 0.763 | 0.763 | 0.763 | 0.762 |
|          | 0.9    | 0.764 | 0.764 | 0.764 | 0.764 |
| T3       | 0.1    | 0.453 | 0.470 | 0.542 | 0.555 |
|          | 0.2    | 0.624 | 0.679 | 0.622 | 0.682 |
|          | 0.3    | 0.654 | 0.734 | 0.649 | 0.735 |
|          | 0.4    | 0.654 | 0.754 | 0.664 | 0.757 |
|          | 0.5    | 0.654 | 0.760 | 0.664 | 0.760 |
|          | 0.6    | 0.654 | 0.760 | 0.661 | 0.760 |
|          | 0.7    | 0.654 | 0.760 | 0.683 | 0.760 |
|          | 0.8    | 0.703 | 0.760 | 0.721 | 0.760 |
|          | 0.9    | 0.758 | 0.760 | 0.758 | 0.760 |

Table 9: Average $MRR$ for LW and $\text{DP}_{0.1\epsilon}$ over T4 and T5. Statistically significant differences between LW and DP are marked in bold.

| Taxonomy | Budget | Uniform Cost | Random Cost |
|----------|--------|--------------|-------------|
|          |        | LW | 10$^6$ | LW | 10$^6$ |
| T4       | 0.1    | 0.527 | 0.523 | 0.547 | 0.530 |
|          | 0.2    | 0.606 | 0.576 | 0.609 | 0.576 |
|          | 0.3    | 0.624 | 0.605 | 0.636 | 0.610 |
|          | 0.4    | 0.644 | 0.623 | 0.644 | 0.630 |
|          | 0.5    | 0.646 | 0.642 | 0.646 | 0.641 |
|          | 0.6    | 0.646 | 0.646 | 0.646 | 0.646 |
|          | 0.7    | 0.646 | 0.648 | 0.646 | 0.646 |
|          | 0.8    | 0.648 | 0.649 | 0.646 | 0.646 |
|          | 0.9    | 0.648 | 0.649 | 0.646 | 0.646 |
| T5       | 0.1    | 0.527 | 0.523 | 0.547 | 0.530 |
|          | 0.2    | 0.606 | 0.576 | 0.609 | 0.576 |
|          | 0.3    | 0.634 | 0.605 | 0.636 | 0.610 |
|          | 0.4    | 0.644 | 0.623 | 0.644 | 0.630 |
|          | 0.5    | 0.646 | 0.642 | 0.646 | 0.641 |
|          | 0.6    | 0.646 | 0.646 | 0.646 | 0.646 |
|          | 0.7    | 0.646 | 0.648 | 0.646 | 0.646 |
|          | 0.8    | 0.646 | 0.649 | 0.646 | 0.646 |
|          | 0.9    | 0.646 | 0.649 | 0.646 | 0.646 |
### 7.4 Efficiency of Proposed Algorithms

Because the efficiency of LW and DP do not depend on any specific cost model, we analyze their efficiencies using uniform cost over the larger taxonomies, i.e., T4 and T5. Table 10 shows the average running time of LW and DP for T4 and T5 over budgets 0.1 to 0.9 using the scaling factor $\epsilon$ of 0.05, 0.1, 0.2 and 0.3 for DP. Both LW and DP, with a reasonably small value of $\epsilon$, $\epsilon \geq 0.1$, are efficient for a design-time task. Overall, LW is more efficient than DP, but DP is almost as efficient as LW when $\epsilon > 0.2$. Both algorithms take longer to run over larger taxonomies, with the exception of $\epsilon = 0.05$ for DP whose reason we explain later in this section. Also, DP takes longer to run as the value of $\epsilon$ becomes smaller. These observations confirm our theoretical analysis of the time complexities of these algorithms. The running time of DP significantly increases as the value of $\epsilon$ changes from $\epsilon = 0.1$ to $\epsilon = 0.05$. Because the size of the matrix required in DP algorithm becomes substantially large for the case of $\epsilon = 0.05$, it occupies most of the available main memory and significantly slows down the program. Also, Java garbage collector spends a lot of time on managing available memory and causes the program to run even more slowly.

Interestingly, DP with $\epsilon = 0.05$ is faster on T5 than on T4. After scaling $u$ and $d$ values in DP algorithm, we remove the concepts with $u$ or $d$ equal to 0 because these concepts will not increase the queriability of any conceptual design. The distribution of $u$ values in T5 is very skewed and has a long tail of concepts with very small $u$ values. Hence, the popularity of many of these concepts will be 0 after scaling. The difference between T4 and T5 in the number of concepts with popularity of 0 after scaling is more for smaller value of $\epsilon$. Using a small value of $\epsilon$ for scaling, T5 will have more such concepts. As T4 has more concepts with non-zero popularities than T5, DP takes longer to run over T4 than T5 for $\epsilon = 0.05$. Table 11 shows the effectiveness of conceptual designs returned by DP for different values of $\epsilon$. Overall, we observe that the effectivenesses of the designs returned by DP consistently improves by reducing the value of $\epsilon$. These results also indicate that DP delivers effective designs using reasonably large values of $\epsilon$, therefore, it can be effectively and efficiently used over large taxonomies.

### 8 Conclusion and Future Work

Annotating entities in large unstructured or semi-structured data sets improves the effectiveness of answering queries over these data sets. It takes significant amounts of financial and computational resources and/or manual labor to annotate entities of a concept. Because an enterprise normally has limited resources, it has to choose a subset of affordable concepts in its domain of interest for annotation. In this paper, 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 an efficient approximation algorithm, called level-wise algorithm, and an exact algorithm with pseudo-polynomial running time for 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. We showed that our formalization framework effectively estimates the amount by which a design improves the effectiveness of answering queries through extensive experiments over real-world datasets, taxonomies, and queries. Our empirical studies also indicated that our algorithms are efficient.

| Taxonomy | Average Running Time (minute) |
|----------|-----------------------------|
|          | LW | DP (0.05) | DP (0.1) | DP (0.2) | DP (0.3) |
| T4       | 0.1 | 0.220 | 0.225 | 0.266 | 0.202 |
|          | 0.2 | 0.251 | 0.250 | 0.247 | 0.242 |
|          | 0.3 | 0.264 | 0.261 | 0.261 | 0.267 |
|          | 0.4 | 0.277 | 0.278 | 0.282 | 0.287 |
|          | 0.5 | 0.291 | 0.291 | 0.291 | 0.291 |
|          | 0.6 | 0.291 | 0.291 | 0.291 | 0.291 |
|          | 0.7 | 0.291 | 0.291 | 0.291 | 0.291 |
|          | 0.8 | 0.292 | 0.292 | 0.292 | 0.292 |
|          | 0.9 | 0.292 | 0.292 | 0.292 | 0.292 |

| T5       | 0.1 | 0.208 | 0.210 | 0.211 | 0.200 |
|          | 0.2 | 0.220 | 0.222 | 0.221 | 0.221 |
|          | 0.3 | 0.223 | 0.233 | 0.233 | 0.233 |
|          | 0.4 | 0.239 | 0.239 | 0.239 | 0.239 |
|          | 0.5 | 0.246 | 0.246 | 0.246 | 0.246 |
|          | 0.6 | 0.247 | 0.247 | 0.247 | 0.247 |
|          | 0.7 | 0.248 | 0.248 | 0.248 | 0.248 |
|          | 0.8 | 0.248 | 0.248 | 0.248 | 0.248 |
|          | 0.9 | 0.248 | 0.248 | 0.248 | 0.248 |

Table 10: Average running time of LW and DP with different values of $\epsilon$ over T4 and T5

| Taxonomy | Budget | DP (0.05) | DP (0.1) | DP (0.2) | DP (0.3) |
|----------|--------|-----------|----------|----------|----------|
| T4       | 0.1    | 0.220     | 0.225    | 0.266    | 0.202    |
|          | 0.2    | 0.251     | 0.250    | 0.247    | 0.242    |
|          | 0.3    | 0.264     | 0.261    | 0.261    | 0.267    |
|          | 0.4    | 0.277     | 0.278    | 0.282    | 0.287    |
|          | 0.5    | 0.291     | 0.291    | 0.291    | 0.291    |
|          | 0.6    | 0.291     | 0.291    | 0.291    | 0.291    |
|          | 0.7    | 0.291     | 0.291    | 0.291    | 0.291    |
|          | 0.8    | 0.292     | 0.292    | 0.292    | 0.292    |
|          | 0.9    | 0.292     | 0.292    | 0.292    | 0.292    |
| T5       | 0.1    | 0.208     | 0.210    | 0.211    | 0.200    |
|          | 0.2    | 0.220     | 0.222    | 0.221    | 0.221    |
|          | 0.3    | 0.223     | 0.233    | 0.233    | 0.233    |
|          | 0.4    | 0.239     | 0.239    | 0.239    | 0.239    |
|          | 0.5    | 0.246     | 0.246    | 0.246    | 0.246    |
|          | 0.6    | 0.247     | 0.247    | 0.247    | 0.247    |
|          | 0.7    | 0.248     | 0.248    | 0.248    | 0.248    |
|          | 0.8    | 0.248     | 0.248    | 0.248    | 0.248    |
|          | 0.9    | 0.248     | 0.248    | 0.248    | 0.248    |

Table 11: Average $p^{\leq 3}$ of DP using different values of $\epsilon$ over T4 and T5
for a design-time task with pseudo-polynomial algorithm delivering more effective designs in most cases.

References
[1] S. Abiteboul, I. Manolescu, P. Rigaux, M. Roussel, and P. Senellart. Web Data Management. Cambridge University Press, 2011.
[2] N. Alon, S. Arora, R. Manokaran, D. Moshkovitz, and O. Weinstein. Inapproximability of densest $\kappa$-subgraph from average-case hardness. http://people.csail.mit.edu/dmoshkov/papers, 2011.
[3] M. Anderson, M. Cafarella, Y. Jiang, G. Wang, and B. Zhang. An Integrated Development Environment for Faster Feature Engineering. In PVLDB, pages 1657–1660, 2014.
[4] M. Anderson et al. Brainwash: A Data System for Feature Engineering. In CIDR, 2013.
[5] P. N. Bennett, K. Svore, and S. T. Dumais. Classification-Enhanced Ranking. In WWW, pages 111–120, 2007.
[6] A. Bhaskara, M. Charikar, A. Vijayaraghavan, V. Guruswami, and Y. Zhou. Polynomial Integrality Gaps for Strong SDP Relaxations of Densest K-subgraph. In SODA, pages 388–405, 2012.
[7] B. Boehm, C. Abts, and S. Chulan. Software Development Cost Estimation Approaches: A Survey. Annals of Software Engineering, 10:177–205, 2000.
[8] S. Chakrabarti, K. Punniyani, and S. Das. Optimizing Scoring Functions and Indexes for Proximity Search in Type-annotated Corpora. In WWW, pages 717–726, 2007.
[9] L. Chiticariu, Y. Li, S. Raghavan, and F. Reiss. Enterprise information extraction: recent developments and open challenges. In SIGMOD, pages 1257–1258, 2010.
[10] J. Chu-Carroll et al. Semantic search via xml fragments: a high-precision approach to ir. In SIGIR, pages 445–452, 2006.
[11] M. Clarka et al. Automatically Structuring Domain Knowledge from Text: An Overview of Current Research. Information Processing & Management, 48(3):552–568, 2012.
[12] N. Dalvi et al. A Web of Concepts. In PODS, pages 1–12, 2009.
[13] X. L. Dong, B. Saha, and D. Srivastava. Less is more: Selecting sources wisely for integration. PVLDB, 6(2):37–48, 2013.
[14] Fabian Suchanek and Gjergji Kasneci and Gerhard Weikum. YAGO: A Core of Semantic Knowledge Unifying WordNet and Wikipedia. In WWW, pages 697–706, 2007.
[15] H. GarciaMolina, J. Ullman, and J. Widom. Database Systems: The Complete Book. Prentice Hall, 2008.
[16] P. Gulhane et al. Web-Scale Information Extraction with Vertex. In ICDE, pages 1209–1220, 2011.
[17] J. Huang and C. Yu. Prioritization of Domain-Specific Web Information Extraction. In AAAI, 2010.
[18] P. Ipeirotis, E. Agichtein, P. Jain, and L. Gravano. To Search or to Crawl? Towards a Query Optimizer for TextCentric Tasks. In SIGMOD, pages 265–276, 2006.
[19] A. Jain, A. Doan, and L. Gravano. Optimizing SQL queries over Text Databases. In ICDE, pages 636–645, 2008.
[20] P. Kanani and A. McCallum. Selecting Actions for Resource-bounded Information Extraction using Reinforcement Learning. In WSDM, pages 253–262, 2012.
[21] S. Khot. Ruling out PTAS for Graph Min-bisection, Densest Subgraph and Bipartite Clique. In FOCS, pages 1025–1071, 2004.
[22] C. Manning, P. Raghavan, and H. Schutze. An Introduction to Information Retrieval. Cambridge University Press, 2008.
[23] A. McCallum. Information Extraction: Distilling Structured Data From Unstructured Text. ACM Queue, pages 48–57, 2005.
[24] J. Pound, I. Ilyas, and G. Weddell. Expressive and Flexible Access to Web-Extracted Data: A Keyword-based Structured Query Language. In SIGMOD, pages 423–434, 2010.
[25] M. Sanderson. Ambiguous Queries: Test Collections Need More Sense. In SIGIR, pages 499–506, 2008.

[26] W. Shen, P. DeRose, R. McCann, A. Doan, and R. Ramakrishnan. Toward Best-Effort Information Extraction. In SIGMOD, pages 1031–1042, 2008.

[27] A. Termehchy, A. Vakilian, Y. Chodpathumwan, and M. Winslett. Which Concepts Are Worth Extracting? In SIGMOD, pages 779–790, 2014.

[28] V. Vazirani. Approximation Algorithms. Springer, 2001.

[29] D. Weiss and B. Tasker. Learning Adaptive Value of Information for Structured Prediction. In NIPS, 2013.

[30] W. Wu, H. Li, H. Wang, and K. Zhu. Probase: A Probabilistic Taxonomy for Text Understanding. In SIGMOD, pages 481–492, 2012.

[31] E. YomTov, S. Fine, D. Carmel, and A. Darlow. Learning to Estimate Query Difficulty. In SIGIR, pages 481–492, 2005.