Which Concepts Are Worth Extracting?

Arash Termehchy
Oregon State University
termehca@oregonstate.edu

Ali Vakilian
MIT
vakilian@mit.edu

Yodsawalai Chodpathumwan,
Marianne Winslett
University of Illinois
ychodpa2,winslett@illinois.edu

ABSTRACT

It is well established that extracting and annotating occurrences of entities in a collection of unstructured text documents with their concepts improve the effectiveness of answering queries over the collection. However, it is very resource intensive to create and maintain large annotated collections. Since the available resources of an enterprise are limited and/or its users may have urgent information needs, it may have to select only a subset of relevant concepts for extraction and annotation. We call this subset a ***conceptual design*** for the annotated collection. In this paper, we introduce the problem of ***cost effective conceptual design***, where given a collection, a set of relevant concepts, and a fixed budget, one likes to find a conceptual design that improves the effectiveness of answering queries over the collection the most. We prove that the problem is generally NP-hard in the number of relevant concepts and propose two efficient approximation algorithms to solve the problem: Approximate Popularity Maximization (***APM*** for short) and Approximate Annotation-benefit Maximization (***AAM*** for short). We show that if there is not any constraints regrading the overlap of concepts, APM is a fully polynomial time approximation scheme. We also prove that if the relevant concepts are mutually exclusive, APM has a constant approximation ratio and AAM is a fully polynomial time approximation scheme. Our empirical results using Wikipedia collection and a search engine query log validate the proposed formalization of the problem and show that APM and AAM efficiently compute conceptual designs. They also indicate that in general APM delivers the optimal conceptual designs if the relevant concepts are not mutually exclusive. Also, if the relevant concepts are mutually exclusive, the conceptual designs delivered by AAM improve the effectiveness of answering queries over the collection more than the solutions provided by APM.

Categories and Subject Descriptors

H.2.1 [**Logical Design**]: Schema and subschema

Keywords

Conceptual design; concept extraction; effective query answering

SIGMOD'14, June 22–27, 2014, Snowbird, UT, USA.

http://dx.doi.org/10.1145/2588555.2610496 .

```
<article>
 ... Michael Jeffrey Jordan is a former American professional
 basketball player ...
</article>
<article>
 ... Michael Jordan is a full professor at the
 University of California, Berkeley ...
</article>
<article>
  ... All six championship teams of Chicago Bulls were led by
  Michael Jordan and Scottie Pippen ...
</article>
```

Figure 1: Wikipedia article excerpts

1. INTRODUCTION

Discovering structured data from large unstructured or semi-structured document collections is an active research area in data management [6]. A popular method of discovering structured data from unstructured or semi-structured documents is *semantic annotation*: extracting the mentions of named entities in a collection and annotating these mentions by their concepts [5, 7, 8, 23]. It is well established that semantically annotating a collection improves the effectiveness of answering queries over the collection [7]. Figure 2 depicts excerpts of semantically annotated Wikipedia articles (*www.wikipedia.org*) whose original and unannotated versions are shown in Figure 1. Since the mentions to the entities named *Michael Jordan* are disambiguated by their concepts, a query interface can deliver more effective results for the queries about *Michael Jordan*, the *scientist*, over the semantically annotated collection than the unannotated one. We call the set of all concepts that have

```
<article>
  ... <athlete> Michael Jeffrey Jordan</athlete> is a former
 <nationality> American </nationality> professional
 basketball player ...
</article>
<article>
  ... <scientist> Michael I. Jordan </scientist> is a full
 professor at the
 <organization>
  University of California, Berkeley
 </organization> ...
</article>
<article>
  ... All six championship teams of
 <organization> Chicago Bulls </organization> were led by
 <athlete> Michael Jordan </athlete> and
 <athlete> Scottie Pippen </athlete> ...
</article>
```

Figure 2: Semantically annotated Wikipedia article excerpts

at least one entity in the collection a *conceptual domain* (*domain* for short). A possible domain for the articles shown in Figure 1 is the set of concepts {*athlete*, *scientist*, *position*, *organization*, *sport*, *nationality*}.

Intuitively, annotating all concepts in a domain will provide more effective results for the input queries. Recent studies, however, indicate that accurately annotating the entities of a concept in a large collection requires developing, deploying, and maintaining complex pieces of software, manual labor, and/or collecting training data, which may take a long time and substantial amount of computational and financial resources [6, 2]. Given a concept, developers have to design and write a program called an *annotator* or *extractor* that finds and annotates all instances of the concept in the collection. One may write hand-tuned programming rules, such as regular expressions, that leverage formatting or language patterns in the text to identify and extract instances of some concepts, such as *Email* [6, 20]. However, formatting and language patterns for most concepts, such as *person* or *protein*, are subtle and involve many exceptions. In these cases, it is not unusual to have thousands of rules to extract concepts [20]. Hence, developing and maintaining annotating programs becomes extremely time-consuming [6].

One may also use machine learning methods for concept extraction [20, 6, 2]. Nevertheless, studies of several recent concept extraction systems show that using and deploying machine learning methods for concept extraction are also very time-consuming and labor-intensive [2, 6]. In order to extract the instances of a concept, developers have to first inspect the data set and identify a set of clues, called *features*, which indicate if some terms in a document refer to an instance of the concept. For instance, the occurrence of word *said* in a sentence may suggest that the subject of the sentence refers to an instance of concept *person*. Then, developers have to write programs to extract these features from the documents. Each concept extractor may use hundreds of features [2, 20]. These efforts are more costly for concepts that are defined in specific domains, such as geology and medicine, as they require extensive collaborations between developers and scarce domain experts. As communication between developers and domain experts is often ineffective, developers may have to spend a great deal of time and sift through the data to find the relevant features [2].

After finding candidate features, developers have to perform *feature selection*: they have to analyze available features, remove some of them, e.g., those that are highly correlated, and select a subset of features that predict the instances of the concept accurately [2, 20]. Developers iterate the steps of finding, extracting, and revising features and testing the annotation program several times in order to create an annotation program with a reasonable accuracy [2]. Finally, since annotation modules need to perform complex text analysis, it may take days or weeks, plus a great deal of computational resources, to execute them over a large collection [15, 24]. Thus, users have to wait a long time for the development and execution of extraction programs before they have a fully annotated collection.

Since the structure and content of documents in many domains evolve over time, the annotation programs should be regularly rewritten and rerun to create an updated annotated collection [12]. Hence, users have to wait a long time for annotation programs to be rewritten and rerun in order to pose their queries over an updated and fully annotated collection [24].

The long delays to create and update fully annotated collections are well recognized as an issue in concept extraction [24, 12]. They are particularly problematic in domains with *urgent* information needs [15, 24]. For example, an FBI agent who needs to query the evolving content of Web sites and social media pages to find and respond to new activities in human trafficking; a stock analyst who has to respond to the changes in stock market in a timely fashion, or an epidemiologist who must act quickly to control the spread of an infectious disease cannot afford to wait for all annotation programs to be (re-)written and (re-)executed [24].

Therefore, an enterprise may decide to select only a subset of the concepts in the domain for annotation or re-annotation, to provide a partially annotated collection relatively quickly. Users can pose their queries over the partially annotated collection and get reasonably effective answers. Moreover, since the available financial or computational resources of most enterprises are limited, they may not be able to hire sufficient number of machine learning experts and acquire computational resources to (re-)write and (re-)execute the annotation programs for all concepts in their domains and select subsets of these domains for annotation. We call this subset of concepts a *conceptual design* (*design* for short) for the annotated collection.

Clearly, an enterprise wants to find a design whose required time (or resources) for annotation does not exceed its limit on turnaround time (or budget) and most improves the effectiveness of answering queries over the annotated collection. Each concept may require different amounts of time and resources for annotating its entities in a collection. For instance, an enterprise may use a freely available and relatively simple annotation program from OpenNLP (*opennlp.apache.org*) to discover the entities of concept *Email*, purchase and deploy a more sophisticated annotation programs from companies, such as *ontotext.com*, to annotated instances of concept *position*, or develop and deploy in-house annotators to identify entities of more domain specific concepts, such as *athlete*. The latter annotators may require more financial resources and/or time to develop and execute than the former one. This scenario suggests a novel conceptual design problem: given a domain and a document collection, we want to find a design for the collection that improves the overall effectiveness of answers to input queries the most, while its annotation costs do not exceed a given budget.

To the best of our knowledge, the choice of a cost effective design for a collection is generally guided only by intuition and has not been studied before. In this paper, we introduce and formalize the problem of *cost effective conceptual design* for semantic annotation. Our formal treatment paves the way for systematic analysis of the problem and shows that intuitively appealing heuristics such as choosing the relatively less costly concepts and/or the ones that appear most often in queries are not generally optimal, even for the cases where all annotators have equal costs. In summary, we make the following contributions:

- We formally analyze the impact of possibly inaccurate annotation of a concept in a collection on the effectiveness of answering queries over the collection. We quantify this impact using a function called Annotation Benefit for two categories of real-world domains: the ones with mutually exclusive concepts and the ones that do not have any constraints regrading the overlap of concepts.
- We introduce and formally define the problem of cost effective conceptual design for semantic annotation as maximizing the value of the Annotation Benefit function over a set of concepts in a domain given a limited time or budget. We prove that the problem over both categories of domains is generally NP-hard in the number of concepts in the domain.
- We propose two efficient approximation algorithms for the problem: Approximate Popularity Maximization (*APM* for short) algorithm and and Approximate Annotation Benefit Maximization (*AAM* for short) algorithm. We prove that the designs returned by APM improve the effectiveness of answering queries

by at most a constant factor less than the optimal design over the domains with mutually exclusive concepts and AAM algorithm is a fully polynomial time approximation scheme in these domains: the effectiveness improvement achieved by its designs will get sufficiently close to the improvement achieved by optimal designs given sufficient running time. We also show that APM is a fully polynomial time approximation scheme for the domains without any constraint regarding the overlap of concepts.

- Our extensive experiments over the collection of Wikipedia articles, concepts from YAGO ontology [23], and queries from the MSN query log [9] show that the Annotation Benefit formula accurately quantifies the impact of a design on the amount of improvement in the effectiveness of answering queries over the annotated collection for both categories of domains.
- Our empirical results indicate that APM finds the optimal designs for most cases where the domain does not have any constraint regrading the overlap of its concepts. They also show that the designs delivered by the AAM algorithm improve the effectiveness of answering queries more than the APM algorithm across domains with mutually exclusive concepts. We also show that both algorithms find these designs in reasonable amounts of times for a preprocessing task.
- Because the complete information about the values of input parameters for AAM may not be available at design time, we explore the sensitivity of this algorithm to the errors in estimating its input parameters and show that when using the input parameters computed over a small sample of the collection, AAM still returns designs that are generally more effective than the ones returned by APM over domains with mutually exclusive concepts.

This paper is organized as follows. Section 2 describes the basic definitions and related works. Section 3 quantifies the impact of a design on the improvement in the effectiveness of answering queries over a collection annotated by the design. Section 4 introduces the problem of cost effective conceptual design and explores its hardness. Section 5 proposes efficient approximation algorithms for the problem and explores their worst-case approximation ratios. Section 6 contains the empirical results about the accuracy of the Annotation Benefit function and the average approximation ratios of the algorithms and Section 7 concludes the paper. A full version of this paper containing proofs and more detailed description of empirical studies can be found at [25].

2. BACKGROUND & BASIC DEFINITIONS

2.1 Related Work

Conceptual design is a topic of research in data management [10]. Our work extends this line of research by introducing and exploring the ability of a conceptual design in effectively answering input queries and its cost-effectiveness.

There is a large body of work on building and optimizing programs that extract entities of a given concept, and systems that manage annotated collections [7, 6, 8, 5, 24, 2]. We build on this work by offering a new pre-processing design phase that can be followed by and coupled with any of these previously proposed approaches. Researchers have proposed several techniques to reduce the execution time of SQL queries over existing databases whose information comes from concept and relation extraction programs [13, 15]. Similar systems optimize the use of information extraction programs to add missing data values to an existing database [16]. These techniques generally improve execution time or storage capacity by processing only the "promising" documents in the collection that contain the information about the database relations, instead of the whole collection. Our work differs in addressing the issues raised at design time rather than query time. We also consider ranking queries as opposed to SQL queries. Our model covers other types of costs in annotation in addition to running-time and storage space. Moreover, we explore using both structured data (i.e. annotated documents) and unstructured data (i.e. unannotated documents) in effectively answering queries.

The authors in [18] have proposed a cost effective methodology for semi-automatic and concurrent construction of ontologies and annotated XML schemas in Web service interfaces. Our work differs in studying the problem of automatically choosing a cost effective set of concepts from a domain for automatic annotation of a collection.

2.2 Basic Definitions

Each concept is a set of named entities (entities for short). Some examples of concepts are *person*, *location*, and *organization*. *Abraham Lincoln* is an entity of concept *person* and *Lincoln Square* is an entity of concept *location*. There may be several mentions of an entity in a collection. For example, *Michael Jeffrey Jordan* and *Michael Jordan* refer to the famous athlete in the collection shown in Figure 2. We call these mentions, *instances* of the entity and for brevity also the instances of its concept (*athlete*).

A domain may contain some constraints on the relationship between its concepts [1]. Concepts C_1 and C_2 are *mutually exclusive* if and only if no entity belongs to both C_1 and C_2. For instance, concepts *person* and *location* are *mutually exclusive*, as no entity is both a person and a location. Our study of real world ontologies, such as DBPedia (*wiki.dbpedia.org/Ontology*), Schema.org (*schema.org*), and YAGO indicates that mutually exclusive concepts appear frequently in these ontologies. For example, in Schema.org each entity should belong to only one of the concepts of *Action*, *BroadcastService*, *Class*, *CreativeWork*, *Event*, *Intangible*, *MedicalEntity*, *Organization*, *Person*, *Place*, *Product*, and *Property*, which are mutually exclusive. As another example, in DBPedia different types of organizations, places, events, devices, and creative works are described by mutually exclusive concepts. Mutually exclusive concepts are also easier to annotate via learning based methods, as one can use the positive training examples of one concept as negative training examples for other concepts [21]. When this constraint is available in the domain, we exploit it to find the cost effective conceptual designs. Concepts in a domain may have other types of relationships such as a subclass/superclass relationship (e.g. person and scientist). Analyzing and solving the problem of cost-effective conceptual design for concepts with other types of relations is a larger undertaking and provides interesting subjects for future work.

The function w_{CO} maps each concept to a real number that reflects the amount of resources used to annotate instances of concept C over collection CO. When the collection is clear from the context, we simply denote the cost function as w. The resources required for concept annotation may include the amount of time, money, or manual labor spent on developing, training, or maintaining an annotator for C, or the computational resources and time to run the annotator over the collection. For example, an enterprise may use the amount of money need to purchase annotators from other companies (e.g. *ontotext.com*) as a cost function. It may also predict the cost of annotator programs that are developed in house using current techniques for predicting costs of software development and maintenance [4]. Researchers have developed effective methods to estimate the execution times of concept extractors [15]

In the absence of any evidence, one may assume that all concepts require equal amount of resources for annotation. As shown in the full version of this paper, it is still challenging to find cost effective designs in this setting [25].

We assume that annotating certain concepts does not impact the cost and accuracies of other concepts in the collection. The costs of (re-)writing, (re-)running, and maintaining an extractor for a concept are still considerable in most cases after coupling its extraction with other related concepts. For example, programmers have to find, extract, and select a great deal of the relevant features for each concept separately.

In this paper, we consider queries that seek information about named entities [7]. Each query $Q : (C, T)$ consists of the concept of the referred-to entity C and a set of keywords T, which describe the referred entity. Some examples of queries are *(person, { Jordan })* or *(location, { Jordan attractions })*. This type of query has been widely used to search annotated collections [7, 11]. Empirical studies on real world query logs indicate that the majority of entity centric queries refer to a single entity [22]. Since this paper is the starting effort to address the problem of cost effective conceptual design, it is reasonable to start with the aforementioned class of queries. Considering more complex queries that seek information about relationships between several entities requires more sophisticated models and algorithms, and therefore, it will require a series of papers and is an interesting topic for future work.

3. THE BENEFIT OF A CONCEPTUAL DESIGN

3.1 Objective Function

Let $\mathcal{S}$ be the design of annotated collection CO and $\mathcal{Q}$ be a set of queries over CO. We would like to quantify the degree by which $\mathcal{S}$ improves the effectiveness of answering queries in $\mathcal{Q}$ over CO. The value of this function should be larger for the designs that help the query interface to answer a larger number of queries in $\mathcal{Q}$ more effectively. It has been shown that most information needs over annotated collections are precision-oriented [8, 7]. Thus, we choose the standard metric of *precision at k* ($p@k$ for short) to measure the effectiveness of answering queries over an annotated collection [19]. The value of $p@k$ for a query is the ratio of the number of relevant answers in the top k returned answers for the query, divided by k. Precision at k has also a simpler form than other precision oriented metrics, such as Mean Reciprocal Rank (MRR) or Normalized Discounted Cumulative Gain (NDCG), thus, it is easier to optimize [19]. We average the values of $p@k$ over queries in $\mathcal{Q}$ to measure the amount of effectiveness in answering queries in $\mathcal{Q}$.

3.2 Effectiveness Improvement for Queries of Annotated Concepts

Let $Q : (C, T)$ be a query in $\mathcal{Q}$. If C is annotated, i.e. $C \in \mathcal{S}$, the query interface will find and return only the documents that contain information about entities in C. It will then rank them according to its ranking function, such as the traditional TF-IDF scoring methods or learning to rank techniques [19]. Our model is orthogonal to the method used to rank the candidate answers. Annotating C in CO will help the query interface avoid non-relevant results that otherwise may have been placed in the top k answers for Q. We call the fraction of queries in $\mathcal{Q}$ whose concept is C the *popularity* of C in $\mathcal{Q}$. Let $u_{\mathcal{Q}}$ be the function that maps concept C to its popularity in $\mathcal{Q}$. When $\mathcal{Q}$ is clear from the context, we simply use u instead of $u_{\mathcal{Q}}$. The portion of queries for which the query interface returns only the documents about entities in their desired concepts is $\sum_{C \in \mathcal{S}} u(C)$. Given all other conditions are the same, the larger the value of $\sum_{C \in \mathcal{S}} u(C)$ is, the more likely it is that the query interface will achieve a larger $p@k$ value over queries in $\mathcal{Q}$. Hence, we may use $\sum_{C \in \mathcal{S}} u(C)$ to compare the degrees of improvement in the value of $p@k$ over queries in $\mathcal{Q}$ achieved by various designs.

Annotators, however, may make mistakes in identifying the correct concepts for occurrences of entities in a collection [7]. An annotator may annotate some appearances of entities from concepts other than C as the occurrences of entities in C. For instance, the annotator of concept *person* may annotate *Lincoln Building* as a person. The *accuracy* of annotating concept C over CO is the number of correct annotations of C divided by the number of all annotations of C in CO. We denote the accuracy of annotating C over CO as $\mathrm{pr}_{CO}(C)$. When CO is clear from the context, we show $\mathrm{pr}_{CO}(C)$ as $\mathrm{pr}(C)$. Given query $Q : (C, T)$ and $C \in \mathcal{S}$, it is reasonable to assume that $1 - \mathrm{pr}(C)$ of the top k results may contain information about entities that do not belong to C. Hence, we should refine our estimate to:

$$\sum_{C \in \mathcal{S}} u(C)\mathrm{pr}(C) \tag{1}$$

in order to reward the designs whose concepts are annotated more accurately.

3.3 Effectiveness Improvement for Queries of Unannotated Concepts

Given query $Q : (C, T) \in \mathcal{Q}$, if $C \notin \mathcal{S}$, there is insufficient meta-data information in the collection for the query interface to identify the occurrences of the entities in C. Therefore, it may view the concept name C and the keywords in T as a bag of words and use some document ranking function to return the top k answers for Q. We like to estimate the fraction of the results for Q that contain 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 $p@k$ value for Q. Based on the available constraints on the relations between concepts in the domain, we provide two different estimations of the fraction of the results for Q that contain a matching entity in concept C.

Domains with mutually exclusive concepts: If the concepts in the domain are mutually exclusive, the annotated concepts may help the query interface to eliminate some non-relevant answers from its results for Q. For example, assume that the instances of concept *location* are annotated and the instances of concept *person* are not annotated in the collection. As these concepts are mutually exclusive, given query *(person, { Jordan })*, query interface can ignore matching instances like *Jordan River* for this query. Because text documents are coherent, they do not usually contain information about entities with similar or the same name but from mutually exclusive concepts. For instance, it is unlikely to find a document that contains information about both *Jaguar*, the vehicle, and *Jaguar*, the animal. Hence, the query interface can eliminate the candidate answers for Q whose matched terms are annotated by concepts other than the concept of Q. By removing these non-relevant answers from its ranked list, the query interface may improve the value of $p@k$ for Q.

In order to compute the fraction of candidate answers for Q whose matching instances belong to C, we have to first calculate the fraction of candidate answers that survive the elimination. This ratio, however, may vary across different queries in $\mathcal{Q}$ as some queries may have more candidate answers with matched annotated instances from concepts in $\mathcal{S}$ than others. Estimating this ratio per

query is generally hard as it may require estimating and computing model parameters per query. Particularly, detailed information about queries in a query workload such as their candidate answers may not be always available. Hence, in order to have an estimation which can be efficiently and effectively computed over a large number of queries, we assume that all queries in $\mathcal{Q}$ have equal ratios of candidate answers that contain matched instances of a certain concept in the domain. We, further, estimate this ratio for the concept by the fraction of documents in the collection that contain instances of the concept. Our empirical results using queries from a real world search engine query log and collection, which are reported in Section 6, show that in spite of these simplifying assumptions, our model effectively estimates the degrees of improvement achieved by various designs for a collection.

Let $d_{CO}(E)$ denote the fraction of documents that contain instances of concept E in collection CO. These instances may or may not be annotated depending on whether $E \in \mathcal{S}$. We call $d_{CO}(E)$ the *frequency* of E over CO. When CO is clear from the context, we denote the frequency of E as $d(E)$. Given design $\mathcal{S}$ for collection CO, we want to compute the fraction of the candidate answers for query $Q : (C, T)$ that contain a matching instance of concepts $E \notin \mathcal{S}$. In order to simplify our model, we estimate this fraction as $\sum_{E \notin \mathcal{S}} d(E)$. Our experimental results in Section 6.2 indicate that in spite of this simplification, our objective function effectively captures the degree of improvement delivered by a conceptual design over a collection. This portion of answers will stay in the list of results for Q after the query interface eliminates all candidate answers with matching instances from concepts in $\mathcal{S}$. Hence, the fraction of the candidate answers that contain a matching instance of concept C in the list of answers for a query in $\mathcal{Q}$ is $\frac{d(C)}{\sum_{E \notin \mathcal{S}} d(E)}$. Using this estimation and equation 1, we formally define the function that estimates the likelihood of improvement for the value of $p@k$ for both queries that belong and queries that do not belong to the conceptual design in a query workload over a collection that is annotated by concepts in design $\mathcal{S}$.

DEFINITION 3.1. *Given domain $\mathcal{C}$ with mutually exclusive concepts, query workload $\mathcal{Q}$, and conceptual design $\mathcal{S} \subseteq \mathcal{C}$, the annotation benefit of $\mathcal{S}$ is:*

$$AB(\mathcal{S}) = \sum_{C \in \mathcal{S}} u(C)\mathrm{pr}(C) + \sum_{C \notin \mathcal{S}} u(C) \frac{d(C)}{\sum_{E \notin \mathcal{S}} d(E)}. \quad (2)$$

Overall, the annotation benefit estimates the likelihood in improving users' satisfaction by answering queries more precisely. The larger the value of the annotation benefit is for design $\mathcal{S}$ over collection CO, the more likely it is that $\mathcal{Q}$ will have a larger average $p@k$ over the version of CO annotated by concepts in $\mathcal{S}$.

The first term of the annotation benefit in equation 3.1 reflects the portion of queries for which the query interface returns only the candidate answers with instances matching to the concept of the query. It is larger for the concepts that are more frequently used in queries. For example, let a domain contain concepts C_1, C_2, and C_3 where the instances of C_1 appear in 90% of queries and 1% of documents, the instances of C_2 occur in 1% of queries and 90% of documents, and the instances of C_3 appear in 9% of queries and 9% of documents. If all annotators have perfect accuracies (i.e. $pr(C)= 1$, $C \in \{C_1, C_2, C_3\}$), we have $\sum_{C \in \{C_1\}} u(C) > \sum_{C \in \{C_2\}} u(C)$. Although C_1 appears in only 1% of documents, it is used in 90% of queries. Hence, it is more likely that the query interface will answer the input queries more effectively if we annotate the instances of C_1 rather than C_2 in the collection.

The second term represents the impact of annotating the concepts in $\mathcal{S}$ on the likelihood of improving the precision of answering queries whose concepts are not in $\mathcal{S}$. Given that the concept of a query does not belong to $\mathcal{S}$, the more frequent the concepts in $\mathcal{S}$ in the collection are, the more non-relevant answers the query interface can eliminate.

Domains without constraints regarding the overlap of concepts: If there is not any constraint on the relations between concepts in the domain, e.g. whether they are mutually exclusive or superclass/subclass, the query interface has to examine all documents in the collection to answer Q. For example, assume that a domain contains concepts *actress* and *director* and the entities of *actress* are annotated in the collection. Given query *(director, { Rossellini })*, the query interface cannot filter out its matching instances from concept *actress* like *Isabella Rossellini* because concepts *actress* and *director* are not mutually exclusive. Thus, if the instances of concept C are not annotated, $C \notin \mathcal{S}$, the fraction of candidate answers of $Q : (C, T)$ that contain 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 $p@k$ for all queries in a query workload over a collection that is annotated by concepts in design $\mathcal{S}$ over domains without any constraint.

DEFINITION 3.2. *Given domain $\mathcal{C}$ without any constraint, query workload $\mathcal{Q}$, and conceptual design $\mathcal{S} \subseteq \mathcal{C}$, the annotation benefit of $\mathcal{S}$ is:*

$$AB(\mathcal{S}) = \sum_{C \in \mathcal{S}} u(C)\mathrm{pr}(C) + \sum_{C \notin \mathcal{S}} u(C)d(C). \quad (3)$$

Similar to the formula for annotation benefit in equation 2, the first term of the annotation benefit in equation 3 reflects the group of queries for which the query interface returns only the candidate answers with instances matching to their concepts. The second term of the annotation benefit in equation 3, however, is different from the second term in equation 2 and represents the impact of the frequency of a concept that is not in $\mathcal{S}$ on the likelihood of the precisions of its queries.

Some domains may contain a mix of mutually exclusive and overlapping concepts. Analyzing and solving the problem of cost-effective conceptual design for such domains is a larger undertaking, which requires more space than one paper and provides an interesting subject for future work.

3.4 Estimating Input Parameters

According to Definitions 3.1 and 3, we need popularities, frequencies, and accuracies of annotation for each concept in a domain, in order to compute the Annotation Benefit of a design over the domain. These values, however, are not usually available before annotating the instances of concepts in the collection. Similar issues arise in database query optimization, where the complete information about running times of operators in a query are not available before running the query [10].

Our empirical results indicate that it is sufficient to compute popularities, frequencies, and accuracies of annotating concepts over only a sample of query workload or collection (e.g. 384 out of about 1 million documents) in order to effectively estimate the values of Annotation Benefit function for designs in a domain. The enterprise may use methods such as crowd sourcing to compute the popularities and frequencies of concepts over such small samples. These annotated documents may be also used as training data for the annotation program of the concept if it is selected for annotation. In some settings, a sample workload of queries with their

concepts is not available, i.e. we may have access only to pure keyword queries. The enterprise can use the click-through information of sample queries to effectively find their associated concepts [3].

An enterprise may use the Annotation Benefit function to choose the concepts for which it should develop annotation programs, therefore, it may not know the accuracies of the annotation programs in design time. Because one has to spend more time and resources to develop a more accurate annotator, the accuracy of annotating a concept somewhat represents the cost of developing its annotation program. Hence, the enterprise may set the accuracy of annotating a concept to a reasonable value that can be achieved using its associated cost. It may also compute and compare the values of Annotation Benefit for designs across multiple assignments of costs and accuracies of annotating concepts in the domain.

4. THE COST EFFECTIVE CONCEPTUAL DESIGN PROBLEM

Since the resources available to develop, maintain, and execute concept annotators are limited, our goal is to find a conceptual design $\mathcal{S}$ such that annotating the concepts in $\mathcal{S}$ in the queries and collection maximizes the annotation benefit. For each concept $C \in \mathcal{C}$, we define the cost of annotation to reflect the amount of resources required to annotate instances of the concept C in the collection. Let B denote the amount of resources available to perform the annotation. Annotating a set of concepts $\mathcal{S}$ is feasible if $\sum_{C\in\mathcal{S}} w(C) \leq B$. We formally define the annotation benefit problem as follows.

PROBLEM 4.1. *Given a domain $\mathcal{C}$, the goal of the* COST EFFECTIVE CONCEPTUAL DESIGN *problem is to construct a conceptual design $\mathcal{S}$ that maximizes the annotation benefit (AB) while satisfying the constraint $w(\mathcal{S}) \leq B$.*

In the case of domains with no constraints, we can rewrite the annotation benefit function as follows.

$$\begin{aligned} AB(\mathcal{S}) &= \sum_{C\in\mathcal{S}} u(C)\mathrm{pr}(C) + \sum_{C\notin\mathcal{S}} u(C)d(C) \\ &= \sum_{C\in\mathcal{S}} u(C)(\mathrm{pr}(C) - d(C)) \\ &\quad + \sum_{C\in\mathcal{C}} u(C)d(C). \end{aligned}$$

where the term $\sum_{C\in\mathcal{C}} u(C)d(C)$ is independent of $\mathcal{S}$. Since concept annotation is an informative process, i.e. it annotates concepts more effectively than a random algorithm, we have $\mathrm{pr}(C) - d(C) > 0$. Thus, the optimization problem in this setting is following.

$$\max \quad \sum_{C\in\mathcal{S}} u(C)(\mathrm{pr}(C) - d(C)), \text{ s.t.} \sum_{C\in\mathcal{S}} w(C) \leq B$$

Provided that we have n concepts in our domain, the COST EFFECTIVE CONCEPTUAL DESIGN problem over domains with no constraints is essentially the same as the 0-1 KNAPSACK problem with n objects, where the value of each object O_C is $u(C)(\mathrm{pr}(C) - d(C))$ and its weight is $w(C)$. Since 0-1 KNAPSACK problem is NP-hard, the the COST EFFECTIVE CONCEPTUAL DESIGN problem over domains with no constraints is also NP-hard.

We prove that the COST EFFECTIVE CONCEPTUAL DESIGN problem is NP-hard for domains with mutually exclusive concepts by a reduction from the following NP-complete variant of the PARTITION problem [17].

PROBLEM 4.2. *Let $\mathcal{A} = \{a_1, \ldots, a_{2m}\}$ be a set of $2m$ positive integers that sum up to $2A$, such that for each $a \in \mathcal{A}$, $\frac{A}{m+1} < a < \frac{A}{m-1}$. The goal of this problem is to decide whether there exists a set $\mathcal{I} \subset \mathcal{A}$ such that $\sum_{a\in\mathcal{I}} a = A$.*

THEOREM 4.3. *The* COST EFFECTIVE CONCEPTUAL DESIGN *problem over a domain with mutually exclusive concepts is NP-hard.*

In the following section, we design efficient approximation algorithms for the COST EFFECTIVE CONCEPTUAL DESIGN problem.

5. APPROXIMATION ALGORITHMS

A brute force algorithm for the COST EFFECTIVE CONCEPTUAL DESIGN problem may take several days or weeks of computation even if the domain contains only 50 concepts. In this section, we design some efficient approximation algorithms for the COST EFFECTIVE CONCEPTUAL DESIGN problem. Consider a maximization problem $\mathcal{M}$. A polynomial time algorithm $\mathcal{A}$ is an α-approximation to $\mathcal{M}$ if $\mathrm{SOL}_{\mathcal{A}} \geq \frac{1}{\alpha}\mathrm{OPT}_{\mathcal{M}}$, where $\mathrm{SOL}_{\mathcal{A}}$ is the value of the solution returned by $\mathcal{A}$ and $\mathrm{OPT}_{\mathcal{M}}$ is the value of the optimal solution to $\mathcal{M}$.

5.1 Approximate Popularity Maximization Algorithm (APM)

We can use available efficient algorithms with bounded approximation ratios for the 0-1 KNAPSACK problem to solve the COST EFFECTIVE CONCEPTUAL DESIGN problem over domains without any constraint with the same approximation ratios. This algorithm will solve a 0-1 KNAPSACK problem for concepts in the domain, where the value of each concept is $u(C)(\mathrm{pr}(C) - d(C))$ and its weight is $w(C)$. An algorithm for a maximization problem is a *Fully polynomial time approximation scheme (FPTAS)* if its running time is polynomial in the size of the input and $(1/\varepsilon)$ and its approximation ratio is $(1+\varepsilon)$ for a given $0 < \epsilon$. Since 0-1 KNAPSACK problem is NP-hard, FPTAS is the best possible approximation for the problem, unless $P = NP$ [14]. In our experiments, we consider an FPTAS algorithm of 0-1 KNAPSACK problem described in [14] that uses a dynamic programming approach. Thus there is an FPTAS algorithm for COST EFFECTIVE CONCEPTUAL DESIGN problem over domains with no constraint.

Moreover, we use the idea behind FPTAS algorithms of 0-1 KANPSACK problem to devise an algorithm for the COST EFFECTIVE CONCEPTUAL DESIGN problem over domains with mutually exclusive concepts. This algorithm ignores the improvement in effectiveness of answering the queries whose concepts are not in the design of a collection. As discussed in Section 3, this improvement is achieved by eliminating the non-relevant answers whose concepts are in the design of the collection from the list of candidate answers for these queries. This degree of improvement is represented by the second term of the annotation benefit function. Hence, this algorithm picks a conceptual design $\mathcal{S}$ with maximum value of $\sum_{C\in\mathcal{S}} u(C)\mathrm{pr}(C)$. We call this modified problem the POPULARITY MAXIMIZATION problem. More formally, given a domain $\mathcal{C}$, the POPULARITY MAXIMIZATION problem maximizes $\sum_{C\in\mathcal{S}} u(C)\mathrm{pr}(C)$ subject to $\sum_{C\in\mathcal{S}} w(C) \leq B$.

The following lemma shows that we can design a constant factor approximation algorithm for the COST EFFECTIVE CONCEPTUAL DESIGN problem by applying an algorithm with bounded approximation ratio for the POPULARITY MAXIMIZATION problem.

LEMMA 5.1. *A ρ-approximation algorithm for the* POPULARITY MAXIMIZATION *problem is a $(\rho + 1/\mathrm{pr}_{min})$-approximation*

for the COST EFFECTIVE CONCEPTUAL DESIGN *problem over domains with mutually exclusive concepts, where* $\text{pr}_{min} = \min_{C \in \mathcal{C}} \text{pr}(C)$.

In particular, if $\text{pr}(C) = 1$ for all $C \in \mathcal{C}$, a ρ-approximation of the POPULARITY MAXIMIZATION problem is a $(\rho+1)$-approximation for the COST EFFECTIVE CONCEPTUAL DESIGN problem.

The POPULARITY MAXIMIZATION problem is also a version of 0-1 KNAPSACK problem with n objects, if we choose the value of each object O_C to be $u(C)\text{pr}(C)$ and its weight to be $w(C)$. Thus, in our experiments we use the FPTAS algorithm for Knapsack [14] to solve this problem.

COROLLARY 5.2. *An FPTAS algorithm that returns a* $(1+\varepsilon)$*-approximate solution to the* POPULARITY MAXIMIZATION *problem is a* $(1+\varepsilon+1/\text{pr}_{min})$*-approximation algorithm for the* COST EFFECTIVE CONCEPTUAL DESIGN *problem over domains with mutually exclusive concepts whose running time is polynomial in* $\frac{1}{\varepsilon}$ *and the number of concepts.*

We call this algorithm the Approximate Popularity Maximization (*APM*) algorithm.

5.2 Approximate Annotation Benefit Maximization Algorithm (AAM)

In this section we present an FPTAS algorithm for the COST EFFECTIVE CONCEPTUAL DESIGN problem over the domains with mutually exclusive concepts. Since in Theorem 4.3 we proved that the problem is NP-hard, FPTAS is the optimal approximation guarantee for the problem unless $P = NP$. The algorithm is based on the dynamic programing method of the 0-1 KNAPSACK problem in addition to some scaling techniques. For simplicity in exposition of the algorithm, we assume that $\text{pr}(C) = 1$ for each $C \in \mathcal{C}$. However, in Remark 5.7 we state that our approach works for an arbitrary pr function, given an additional property that usually holds in practice.

Given a fixed constant N, we define the BOUNDED COST EFFECTIVE(N) problem as follows.

$$\max_{S} \quad f(N,S) = \frac{1}{N}(N \sum_{C \in \mathcal{S}} u(C) + \sum_{C \in \mathcal{C}-\mathcal{S}} u(C)d(C)) \quad (4)$$
$$\text{s.t.} \sum_{C \in \mathcal{C}-\mathcal{S}} d(C) \leq N$$
$$\sum_{C \in \mathcal{C}-\mathcal{S}} w(C) \leq B$$

In addition to the cost constraint that we had previously, BOUNDED COST EFFECTIVE(N) has a constraint over frequency of documents. Let $\langle \mathcal{C}, B, d, u, w \rangle$ be an instance of the COST EFFECTIVE CONCEPTUAL DESIGN problem. For any value of N, the value of the optimal solution of the BOUNDED COST EFFECTIVE(N) on $\langle \mathcal{C}, B, d, u, w \rangle$ is not more than the optimal solution of the annotation benefit of COST EFFECTIVE CONCEPTUAL DESIGN on $\langle \mathcal{C}, B, d, u, w \rangle$. Moreover, for a fixed N, the objective function of the COST EFFECTIVE CONCEPTUAL DESIGN is a separable function. Thus it is easier to find the maximum value of BOUNDED COST EFFECTIVE(N) for a fixed N rather than finding the optimal conceptual design of the COST EFFECTIVE CONCEPTUAL DESIGN problem.

LEMMA 5.3. *Let* OPT *be the value of the optimal solution of the* COST EFFECTIVE CONCEPTUAL DESIGN *problem and let* OPT_{bd} *be the maximum value of an optimal solution of* BOUNDED COST EFFECTIVE(N) *over different values of* N. *Then* $\text{OPT} = \text{OPT}_{\text{bd}}$. *Moreover, the same set of concepts (conceptual design) obtains the optimal value in both functions.*

Lemma 5.3 implies that in order to find a set with the maximum annotation benefit we can instead solve BOUNDED COST EFFECTIVE(N) for all different values of N and return the set that obtains the maximum value. In other words, first we give an FPTAS for BOUNDED COST EFFECTIVE(N), where N is a given fixed value. The first step is to check whether for the given N there exists a feasible solution to BOUNDED COST EFFECTIVE(N). For the given N, a feasible solution has to contain all concepts C that $d(C) > N$. Let $\mathcal{S}_{\text{rem}} = \{C | d(C) > N\}$ and $\mathcal{C}_{\text{rem}} = \mathcal{C} - \mathcal{S}_{\text{rem}}$. If $w(\mathcal{S}_{\text{rem}}) > B$, there is no feasible solution for BOUNDED COST EFFECTIVE(N). Otherwise; we select all concepts in $\mathcal{S}_{\text{rem}}$ and we set $B_{\text{rem}} = B - w(\mathcal{S}_{\text{rem}})$ to be the leftover budget. The problem is equivalent to optimize the bounded problem on $\mathcal{C}_{\text{rem}}$, B_{rem} and N. Now, for each $C \in \mathcal{C}_{\text{rem}}$ we have $d(C) \leq N$. To solve BOUNDED COST EFFECTIVE(N) optimally for the given N, we can apply dynamic programming. Let $V_{\text{init}}(N) = \sum_{C \in \mathcal{C}_{\text{rem}}} u(C)d(C)/N$. We can rewrite the objective function of BOUNDED COST EFFECTIVE(N) as follows:

$$\sum_{C \in \mathcal{S}} v(C) + V_{\text{init}}(N)$$

where $v(C) = u(C)(1 - d(C)/N)$ for each $C \in \mathcal{C}_{\text{rem}}$

Let $\mathcal{C}_{\text{rem}} = \{C_1, \cdots, C_n\}$. We define $Q[i, P, X]$ to be the minimum required cost that we must pay to obtain a solution of BOUNDED COST EFFECTIVE(N) of value at least $P - V_{\text{init}}$ if we are only allowed to annotate concepts from the first i concepts of $\mathcal{C}_{\text{rem}}$. We can state the recursive relation of $Q[i, P, X]$ as follows:

1. $Q[0, 0, X] = 0$ for all $0 \leq X \leq N$
2. $Q[0, P, X] = \infty$ for all $P > 0$ and $0 \leq X \leq N$
3. $Q[i, P, X] = \min(Q[i-1, P, X - d(C_i)],$
   $Q[i-1, \min\{P - v(C_i), 0\}, X] + w(C_i))$

To find the optimal solution of BOUNDED COST EFFECTIVE(N), we need to find the maximum value of V such that $Q[n, V, N] \leq B_{\text{rem}}$. The running time of the described dynamic programming is $O(nVN)$ where V is the value of the optimal solution of the bounded problem for the given N. The described dynamic programming is pseudo-polynomial and we can convert it to an FPTAS via scaling techniques.

LEMMA 5.4. *There exists a* $(1+\varepsilon)$*-approximation algorithm to* BOUNDED COST EFFECTIVE(N) *which runs in* $O(Nn^3/\varepsilon)$.

Although we need to satisfy the document frequency constraint of BOUNDED COST EFFECTIVE(N), $\sum_{C \in \mathcal{C}_{\text{rem}} - \mathcal{S}} d(C) \leq N$, for a given N we can allow $\mathcal{S}$ to violate the document frequency constraint by ε; our ultimate goal is to maximize the annotation benefit of COST EFFECTIVE CONCEPTUAL DESIGN problem. Later we show that the value of AB for a $(1+\varepsilon)$-approximate solution of BOUNDED COST EFFECTIVE(N), $\mathcal{S}$, that violates the document frequency constraint by at most ε is comparable to the optimal solution of BOUNDED COST EFFECTIVEfor the given N.

LEMMA 5.5. *There is a* $(1+\varepsilon)$ *approximation algorithm for* BOUNDED COST EFFECTIVE(N) *that violates* $\sum_{C \in \mathcal{C}_{\text{rem}} - \mathcal{S}} d(C) \leq (1+\varepsilon)N$, *and its running time is* $O(n^4/\varepsilon^2)$.

By Lemma 5.5, we have an algorithm that finds a solution $\mathcal{C}_s$ with value at least $(1+\varepsilon)$ times the optimal solution of BOUNDED COST

EFFECTIVE(N). However, $\mathcal{C}_s$ may violate the document frequency constraint by ε. Suppose that $\mathcal{C}_s$ is the set returned by the algorithm after performing the described scaling. Let $N_r = \sum_{C\in\mathcal{C}_s} d(C)$ and let N_s be the value of N for which $\mathcal{C}_s$ is returned in our algorithm (since we allow the algorithm to violate the constraint by ε, $N_r \leq (1+\varepsilon)N_s$). Suppose that $\mathcal{C}_o$ is the set with optimal value of AB. Lemma 5.3 and 5.5 imply that $AB(\mathcal{C}_o) \leq (1+\varepsilon)f(N_s, \mathcal{C}_s)$. Thus $AB(\mathcal{C}_s) = f(N_r, \mathcal{C}_s) \geq f(N_s, \mathcal{C}_s)/(1+\varepsilon) \geq (1/(1+\varepsilon)^2)AB(\mathcal{C}_o) \geq (1-2\varepsilon)AB(\mathcal{C}_o) \geq \frac{1}{(1+2\varepsilon)}AB(\mathcal{C}_o)$ where f is the objective function of the BOUNDED COST EFFECTIVEproblem. By maximizing BOUNDED COST EFFECTIVE(N) over all possible values of N ($0 < N \leq D_{total} = \sum_{C\in\mathcal{C}_{\rm rem}} d(C)$), we can find a $(1+\varepsilon)$-approximation[1] of the COST EFFECTIVE CONCEPTUAL DESIGN problem in $O(D_{\rm total}\frac{n^4}{\varepsilon^2})$.

THEOREM 5.6. *The* COST EFFECTIVE CONCEPTUAL DESIGN *problem admits an FPTAS algorithm.*

Instead of checking all possible values of N which lead to a pseudo-polynomial algorithm, we can instead solve the relaxed version of BOUNDED COST EFFECTIVEfor some specific values of N (which is polynomial in the size of input) and still guarantees a $(1-\varepsilon)$-approximation (in the relaxed version we allow the solution to violate the document frequency constraint by a factor of ε). Consider the set $\mathcal{N} = \{N_1, \cdots, N_p\}$ such that $N_i = D_{\min}(1/(1-\varepsilon))^i$ where $D_{min} = \min_{C\in\mathcal{C}_{\rm rem}} d(C)$ and $N_{p-1} \leq D_{\rm total} \leq N_p$. This implies that $p < \log_{(1-\varepsilon)^{-1}}(D_{\rm total}/D_{min}) + 1 = (\log D_{\rm total} - \log D_{\min})/(-\log(1-\varepsilon)) + 1 < O((\log D_{\rm total})/\varepsilon)$, where the last inequality comes from $-\log(1-\varepsilon) = -\ln(1-\varepsilon)/\ln 2 > \varepsilon/\ln 2$. Thus the number of different values of N we need to examine is polynomial in $\log D_{total}$ and $1/\varepsilon$. Suppose that $(N_o, \mathcal{C}_o)$ is the pair that maximizes f, i.e, OPT $= f(N_o, \mathcal{C}_o)$. Let N_g be the smallest member of $\mathcal{N}$ that is greater than N_o. Note that $(N_g, \mathcal{C}_o)$ is a feasible solution to f and since $N_g > N_o$, $N_g f(N_g, \mathcal{C}_o) > N_o f(N_o, \mathcal{C}_o)$. Thus $f(N_g, \mathcal{C}_o) > (N_o/N_g)$OPT. The solution returned by our algorithm is at least $(1-\varepsilon)$ times the maximum of f for N_g. Since $N_o > N_{g-1}$, $N_o/N_g \geq N_{g-1}/N_g$ and thus $f(N_g, \mathcal{C}_o) \geq (N_{g-1}/N_g)$OPT $\geq (1-\varepsilon)$OPT. This implies the value the maximum value of on N_g is at least $(1-\varepsilon)$OPT $\geq \frac{1}{(1+\varepsilon)}$OPT. This fact along with Lemma 5.5 lead us to obtain an FPTAS algorithm with runtime $O((n^4 \log D_{\rm total})/\varepsilon^3)$. Note that $\log D_{\rm total} \leq \log(nD_{\max}) \leq \log n + \log D_{\max}$ where $D_{\max} = \max_{C\in\mathcal{C}_p} d(C)$ and is polynomial in the size of input. Hence the running time is bounded by $O(n^5/\varepsilon^3)$.

REMARK 5.7. *We assumed that* $\Pr(C) = 1$ *for all* $C \in \mathcal{C}$. *However, our approach also works for a realistic function* pr. *For a given N, we define* $v(C) = u(C)(\Pr(C) - \frac{d(C)}{N})$ *and the proof holds as long as* $v(C)$ *is a positive value for all concepts.*

Table 1 summarizes the time complexities and approximation ratios of the approximation algorithms over domains with mutually exclusive concepts.

Algorithm	Approximation ratio	Running time
APM	$2+\varepsilon$	$O(n^3/\varepsilon)$
AAM	$1+\varepsilon$	$O(n^5/\varepsilon^3)$

Table 1: Approximation ratios and time complexities of approximation algorithms over domains with mutually exclusive concepts.

6. EXPERIMENTS

6.1 Experiment Setting

Domains: To validate the accuracy of the annotation benefit function and the effectiveness of our conceptual design algorithms, we use concepts from YAGO ontology version 2008-w40-2 [23]. YAGO organizes its concepts using IS-A (i.e. parent-child) relationships in a DAG with a single root. We define a *level* as a set of concepts that have the same distance (in terms of the number of edges) from the root of the ontology. Most levels in the DAG generally contain a set of mutually exclusive concepts. We select three domains from this ontology for our experiments. All concepts in each domain are mutually exclusive and have at least one instance in our dataset. *Domain M1* consists of 7 concepts from the third level of the ontology. We further select two larger domains from the YAGO ontology to evaluate the average-case performance ratios and efficiency of our approximation algorithms. *Domain M2* consists of 76 mutually exclusive concepts from the fourth level of the ontology, such as *location* and *event*. Since we would like to evaluate our algorithms over domains with more concrete concepts, we expand some relatively abstract concepts such as *whole* to their descendants on the sixth level of the ontology and created a third domain, called *domain M3*. This domain consists of 87 concepts such as *person* and *animal*. We also select an additional domain, called *N1*, from the fifth level of YAGO with 10 concepts whose concepts are not guaranteed to be mutually exclusive. We use this domain to validate the annotation benefit formula for the domains with no constraint and measure the empirical approximation ratio of APM over these domains.

Dataset: We use a semantically annotated version of the Wikipedia collection that is created from the October 8, 2008 dump of English Wikipedia articles [23]. This collection uses concepts from the YAGO ontology. It contains 2,666,190 Wikipedia articles, of which 1,470,661 are annotated. For each domain, we have selected all documents that contains an annotation of a concept in the domain and created a dataset for that domain. The datasets for domain M1, M2, M3 and N1 contain 525,703, 399,792, 927,848 and 186,952 documents respectively. Each annotation contains a confidence value that indicates the accuracy of the annotation. We have used the average confidence values over all annotations of a concept to compute its annotation accuracy. The accuracies of annotations are between 0.75 and 0.95 for concepts in the selected domains.

Query Workload: We use a subset of the MSN query log whose target URLs are Wikipedia articles [9]. Each query contains between 2 to 6 keywords and has one to two relevant answers. Because the query log does not list the concept behind each query, we adopt an automatic approach to find the concepts associated with the query. Given a domain, for each query we find the concept from the domain whose instance(s) match the query terms in its relevant answers. We ignore the queries that match instances from multiple concepts in their relevant answers as these queries do not comply with our query model. The effectiveness of answering some queries may not be improved from semantically annotating the collection [7].For instance, all candidate answers for a query may contain matched instances of the same concept. In order to reasonably evaluate our algorithms, we have not considered the queries whose rankings are the same over the unannotated version and the fully annotated version (i.e. annotating all concepts in the domain) of the collection. This method leads to collecting 98 (98 unique), 187 (118 unique), 1737 (972 unique), and 199 (138 unique) queries for domain M1, M2, M3, and N1 respectively. We use two-fold cross validation to train the u values for concepts in

[1]We let $\varepsilon' = 2\varepsilon$.

	Frequency-based Cost			Random Cost		
Budget	Oracle	PM	AM	Oracle	PM	AM
0.1	0.146	0.146	0.146	0.190	0.188	**0.190**
0.2	0.207	0.207	0.207	0.208	0.205	**0.208**
0.3	0.218	0.218	0.218	0.216	0.216	0.216
0.4	0.218	0.218	0.218	0.218	0.218	0.218

Table 2: Average $p@3$ for Oracle, PM, and AM over domain M1.

each domain. Because some concepts may not appear in the query workload, we smooth the u values using the Bayesian m-estimate method with smoothing parameter 1 and uniform priors [19].

Retrieval System: We index the datasets using Lucene (*lucene.apache.org*) and use BM25 as the underlying retrieval algorithm [19]. Given a query, we first rank its candidate answers using BM25. Then, we apply the information about the concepts in the query and documents to return the documents whose matching instances have the same concept as the concept of the query or to filter out the non-relevant candidate answers for the query if using domain M1, M2 and M3, as explained in Section 3. We performed our experiments on a Linux server with 250 GB of main memory and two quad core processors. We implemented our retrieval system and optimization algorithms using JAVA 1.7.0_51.

Effectiveness Metric: Most queries in our query workloads have one relevant answer. Hence, we measure the effectiveness of answering queries over the dataset using precision at 3 ($p@3$). We measures the statistical significance of our results using the paired-t-test at a significant level of 0.05. The statistically significant improvements are marked in bold in the reported results.

Cost Metrics: We use two types of costs for concept annotation development and maintenance in our experiments. We hypothesize that the cost of building, running, and maintaining an annotator for a concept may be proportional to its frequency as its instances may appear in more diverse contexts in a collection, which may lead to a larger number of rules/features for an annotator and longer running time. We call this type of cost assignment *frequency-based cost*. We also evaluate our algorithms by assigning randomly generated costs to the concepts in a domain. We call this type of cost assignment *random cost*. We report the average $p@3$ over 40 sets of random costs for each budget. We use a range of budgets between 0 and 1 with step size of 0.1, where 1 means sufficient budget to annotate all concepts in a domain.

6.2 Model Validation

In this section, we investigate whether the Annotation Benefit function accurately estimates the likelihood of improvement in effectiveness of answering queries over annotated collections.

Domain with mutually exclusive concepts: We use three algorithms in this set of experiments. Given complete information about the relevant answers of queries, *Oracle* checks all possible designs in a domain whose costs do not exceed a fixed budget and delivers the design with maximum $p@3$ over all queries. Because the designs returned by Oracle deliver the maximum possible effectiveness for answering queries, we use its results to measure how accurately practical methods predict the amount of improvement in effectiveness of answering queries achieved by a design. *AM* is a brute force algorithm that picks the design with maximum Annotation Benefit over a domain given a fixed budget. An intuitively appealing heuristic for finding a design is to select the concepts that are most queried by users. The *PM* algorithm implements this heuristic. PM is a brute force algorithm that finds the design with the maximum value of $\sum_{C \in \mathcal{S}} u(C)\mathrm{pr}(C)$ over a domain given a fixed budget. Since all these algorithms use exhaustive search methods, running them over a domain with a large number of concepts is not practical. Thus, we evaluate these algorithms only over domain M1. In order to precisely evaluate the estimation accuracy of the Annotation Benefit function, we assume that AM has the exact values of concept frequencies. We will explain how to estimate the frequencies of concepts without fully annotating them in Section 6.3.

Table 2 shows the values of $p@3$ over domain M1 delivered by AM and PM using frequency-based and random costs. We have omitted the results of Oracle, AM, and PM for budgets from 0.5-0.9 in table 2, because their results are the same as the ones for budget 0.4. Since the number of concepts in domain 1 is rather small, there are few feasible solutions that can exhaust the budget, given a modest or large budget. Hence, all algorithms find the same or very similar designs for these budgets over domain M1. The designs returned by Oracle, AM, and PM deliver the same values of $p@3$ for answering queries over all budgets for frequency-based cost. In this setting, AM and PM pick the same designs for all budgets between 0.1 and 0.8. The designs selected by AM and PM are different for budget 0.9. Nonetheless, both designs contain 6 out of 7 available concepts in the domain. Answering queries over an annotated collection that contains annotation for all but one of the concepts in the domain will be as effective as answering queries over the fully annotated collection. Therefore, they both achieve the same values of $p@3$. Further, the cost distribution in the frequency based cost setting is very skewed in domain M1. Since the number of concepts is rather small in domain M1 and the cost distribution is skewed, there are very few feasible solutions that can maximize the objective functions of either AM or PM given a small budget. For example, with budget equal to 0.2, there are only two feasible designs that exhaust the budget and one of them maximizes the objective functions of both AM and PM.

Table 2 shows that the designs produced by AM deliver more effective results for queries than the ones generated by PM for budgets 0.1 - 0.2 using random costs. Since the cost distribution of random costs is not as skewed as the cost distribution for frequency-based costs, there are more feasible solutions for both objective functions than the frequency-based cost setting for small budgets. For example, PM and AM include *causal agent* and *psychological feature*, respectively, in their designs for budget equal to 0.1. These designs are not feasible in the frequency-based cost setting. Since *causal agent* is quite frequent in the collection, the matching instances of this concept appear in most of the top candidate answers for queries with this concept. Hence, AM does only slightly worse than PM in returning answers whose matching instances belong to *causal agent* for queries with this concept. Because AM picks *psychological feature* in its design, it is able to effectively answer the queries from this concept. PM, however, does not pick this concept in its design. Because this concept is not very frequent in the collection, the matching instances of most candidate answers for queries with this concept belong to other concepts. Hence, the PM design returns considerably less effective results for these queries than the AM design. AM returns the same designs as Oracle for budgets 0.1 and 0.2 in the random cost setting. As the budget becomes larger, both algorithms pick almost all useful (relatively popular and/or relatively frequent) concepts. Thus, the ranking qualities provided by the designs from these methods are almost the same for larger budgets.

Domains without constraints regrading the overlap of concepts: We use two algorithms in this set of experiments. *Oracle* is the same algorithm used in validation experiments for domains with mutually exclusive concepts. *AM-N* is a brute force algorithm that picks the design with maximum Annotation Benefit over a do-

	Frequency-based Cost			Random Cost		
Budget	Oracle	AM-N	APM	Oracle	AM-N	APM
0.1	0.191	0.191	0.191	0.173	**0.173**	0.160
0.2	0.229	0.229	0.229	0.205	**0.205**	0.196
0.3	0.238	0.238	0.238	0.230	0.230	0.230
0.4	0.238	0.238	0.238	0.235	0.235	0.235
0.5	0.238	0.238	0.238	0.237	0.237	0.237
0.6	0.238	0.238	0.238	0.238	0.238	0.238

Table 3: Average $p@3$ for Oracle, AM-N, and APM ($\epsilon = 0.001$) over Domain N1.

Domain	Domain M1		Domain M2		Domain M3	
	APM	AAM	APM	AAM	APM	AAM
0.1	0.146	0.146	0.196	**0.230**	0.145	0.145
0.2	0.177	**0.207**	0.203	**0.237**	0.164	0.165
0.3	0.218	0.218	0.205	**0.239**	0.175	0.176
0.4	0.218	0.218	0.203	**0.241**	0.183	**0.196**
0.5	0.218	0.218	0.237	0.241	0.175	**0.198**
0.6	0.218	0.218	0.239	0.241	0.202	0.202
0.7	0.218	0.218	0.241	0.241	0.202	0.202
0.8	0.211	0.218	0.235	0.241	0.202	0.202
0.9	0.218	0.218	0.241	0.241	0.202	0.202

Table 4: Average $p@3$ for AAM (with $\epsilon = 0.1$) and APM (with $\epsilon = 0.001$) using frequency-based costs.

main with no constraint given a fixed budget. Since *PM* heuristic is very similar to *AM-N*, we do not report its results in this section. Table 3 shows the values of $p@3$ over domain N1 delivered by Oracle and AM-N using frequency-based and random costs. All results for budget 0.7-0.9 are omitted as they are the same as the results over budget 0.6. Note that since the number of concepts in domain N1 is larger than domain M1, this increases the running time of Oracle. Due to limited amount of time, we perform the experiment over domain N1 using 20 sets of random costs.

Overall, ranking quality delivered by AM-N is the same as Oracle. AM-N and Oracle pick the same design for all budgets for frequency-based cost. Intuitively, more popular and accurately annotated concepts should deliver a better ranking quality for domains without any constraint because query interface can use only the annotations for the concept of each query to improve the ranking quality of its answers. As opposed to the domain with mutually exclusive concepts, query interface cannot use annotations of concepts other that the concept in the query to filter non-relevant answers. AM-N generally picks the same designs as Oracle for runs of random costs. AM-N chooses different designs from Oracle in a few runs, but the both designs lead to almost the same amount of improvement in ranking qualities for queries. These results confirms our assumption that Annotation Benefit formula is suited as an objective function for domains with not constraint.

6.3 Effectiveness of Approximation Algorithms

Parameters Estimation: In addition to the popularities (u) of concepts in the query workload, AAM requires the value of the frequency (d) for each concept in the collection. The exact frequency of a concept, however, cannot be determined before annotating all its instances. We estimate the frequencies of concepts using a small sample of randomly selected documents from the collection. For each domain, we calculate the frequency of each concept over a random sample of 384 documents from the collection which corresponds to an estimation error rate of 5% under the 95% confidence level. Similar to computing concepts' popularities, we smoothed the value of d using Bayesian m-estimates with smoothing parameter of 1 and uniform priors.

Domain	Domain M1		Domain M2		Domain M3	
	APM	AAM	APM	AAM	APM	AAM
0.1	0.179	**0.189**	0.221	**0.240**	0.192	**0.202**
0.2	0.201	**0.207**	0.223	**0.240**	0.193	**0.202**
0.3	0.215	0.214	0.226	**0.240**	0.194	**0.202**
0.4	0.218	0.217	0.227	**0.240**	0.195	**0.202**
0.5	0.218	0.218	0.229	**0.241**	0.197	**0.202**
0.6	0.218	0.218	0.231	**0.241**	0.197	**0.202**
0.7	0.218	0.218	0.232	**0.241**	0.198	**0.202**
0.8	0.218	0.218	0.234	**0.241**	0.199	**0.202**
0.9	0.218	0.218	0.237	**0.241**	0.202	0.202

Table 5: Average $p@3$ of AAM (with $\epsilon = 0.3$) and APM (with $\epsilon = 0.001$) using random costs.

Domains with mutually exclusive concepts: Tables 4 and 5 show the values of $p@3$ for the APM and AAM algorithms for all mutually exclusive domains using frequency-based and random costs, respectively. Generally, the designs generated by AAM improve the values of $p@3$ significantly more than the designs produced by APM, over all domains and both types of cost metrics. As discussed in Section 3, the optimal design should balance three types of impacts. First, it should contain the most popular concepts so the query interface can return potentially relevant answers to as many queries as possible. Second, if a concept is very frequent, most candidate answers for queries with this concept contain matching instances of this concept. Hence, if the concept is relatively costly, the optimal design should not include these concepts as they may not worth annotating. Third, it should contain relatively frequent and relatively cheap concepts so that the query interface can eliminate many non-relevant answers from the list of results for the queries whose concepts are not in the design.

In our experiments, the designs produced by APM have larger overall popularities (u values) than the designs selected by AAM, across all domains and cost metrics. We have observed that in general the most popular concepts in users' queries may not be the most frequent ones in the collection. The designs picked by AAM do not normally include the most popular concepts. Instead, they contain a larger number of relatively popular concepts than the designs selected by APM over all domains and cost metrics. Generally, relatively popular concepts are also rather frequent. Since the overall frequencies of the designs produced by AAM are generally larger than the ones selected by APM, they help the query interface to eliminate more non-relevant answers from the results of the queries whose concepts are not included in these designs. Because these designs include relatively popular concepts, they also help the query interface to return the relevant answers to a relatively large number of queries.

In a small number of cases, the designs generated by APM deliver a larger $p@3$ than the ones produced by AAM. Although the differences between AAM and APM in these cases are not statistically significant, it is interesting to explore the reasons behind these improvements. The designs generated by APM for budget 0.3 and 0.4 over domain M1 using random costs deliver larger values of $p@3$ than the designs of AAM. In both budgets, the relative effectiveness improvement of APM over AAM in each budget is due to a single run where APM selects a design with larger value of Annotation Benefit than the design picked by AAM. This illustrates the fact that both methods are approximation algorithms and sometimes they may return quite different answers from their optimal solutions.

Generally, the differences between the values of $p@3$ achieved by the designs of AAM and APM are smaller for larger budgets across all domains and cost metrics. This is mainly due to the

fact that AAM and APM can afford to include most of the popular and frequent concepts in their designs for medium or large budgets. Adding the concepts that are rare in the collection or query workload does not improve the effectiveness of answering queries considerably. Because domain M1 has a relatively small number of concepts, both algorithms pick similar designs given a smaller amount of budget for this domain than other domains.

In some cases, the designs generated by APM deliver smaller values of $p@3$ for larger budgets. For instance, the design for budget 0.8 delivers a smaller value of $p@3$ than the one for budget 0.7 over domain M1 when frequency-based cost is used. Given sufficient budget, APM may replace reasonably popular and frequent concepts with more popular and less frequent concepts. As discussed in Section 3 and the beginning of this section, this may have a negative impact on $p@3$ for answering queries over the annotated collection.

The values of $p@3$ for the designs generated by AAM over domain M2 and domain M3 when random costs are used are almost the same over all budgets greater than 0.1. The distribution of frequencies and popularities of concepts are very skewed in these domains, where relatively small number of concepts (e.g. 10 concepts in domain M2) have a large portion of the total frequency and popularity in the domain. Since the costs are assigned randomly, in most runs AAM is able to pick these concepts using a relatively small budget. AAM adds new concepts to this set given larger budgets. The new concepts, however, do not improve the effectiveness of answering queries over the annotated collection.

APM cannot find the set of more popular concepts given a small or moderate budget. The algorithm used in APM has two main steps [17]. It separates concepts into two sets, popular and unpopular. It then uses a dynamic programming method to find the optimal solution from the set of popular concepts. If there are still some budget left, it greedily picks concepts from the set of unpopular concepts until the budget is exhausted. The decision of how to partition concepts into these two sets is based on an approximation, which is not accurate in many cases. Hence, in some cases the concepts that belong to the optimal design may placed in the set of unpopular concepts. The greedy algorithm used to pick the concepts in the unpopular set sorts them based on the ratio of $\frac{u(C)\times pr(C)}{w(C)}$, where u is the popularity, pr is the accuracy, and w is the cost of concept C, and select the top concepts. This method leaves out some desired concepts that are relatively popular but expensive.

Domains without constraints regrading the overlap of concepts: We investigate the effectiveness of the version of APM algorithm introduced in Section 5.1 for domains without any constraint. Table 3 shows the values of $p@3$ over domain N1 delivered by Oracle, AM-N and APM using frequency-based and random costs. Overall, the results delivered by APM are the same as that of Oracle and AM-N except at budget 0.1 and 0.2 of random cost. APM generally selects the same designs as that of PM. Hence, it delivers the same ranking qualities as the optimal solutions. Since APM is an approximation algorithm, it cannot find solutions that maximizes concept popularity in some runs of random cost. For example, APM picks the design consists of *literary composition* where AM-N picks the design with *series* and *dramatic composition* in one of the runs. The overall popularity of the latter design is larger than the former one. Thus, APM cannot answer queries as effectively as AM-N does. As the costs are quite skewed in frequency-based runs, APM almost always picks the same designs as AM-N in these runs.

6.4 Efficiency of Approximation Algorithms

This section studies the efficiency and scalability of our approximation algorithms. Due to limited space, we show the efficiency and scalability of AAM and APM using only frequency-based costs over domains M1, M2 and M3. Our experiments over the random based costs show similar results for the scalability of the algorithms. Since the running time and memory consumption of APM is similar over the domains with mutually exclusive concepts and the domains without any constraints, we report the scalability results for APM only over the domains with mutually exclusive concepts.

	ϵ	0.5	0.3	0.1	0.01	0.001
AAM	Domain M1	1	2	2	-	-
	Domain M2	1	5	102	-	-
	Domain M3	4	15	128	-	-
APM	Domain M1	1	2	2	2	5
	Domain M2	1	2	2	3	12
	Domain M3	4	14	15	15	23

Table 6: Average running times of AAM and APM (in minutes)

	ϵ	0.5	0.3	0.1	0.01	0.001
AAM	Domain M1	348	492	635	-	-
	Domain M2	1667	6498	84139	-	-
	Domain M3	1326	5608	63466	-	-
APM	Domain M1	184	184	184	215	1976
	Domain M2	184	184	184	215	4933
	Domain M3	184	184	184	471	7732

Table 7: Average memory usages of AAM and APM (in MB)

Efficiency: Table 6 shows the average running time of APM and AAM algorithms over domain M1, M2, and M3 with budget 0.1 to 0.9 using various several values between 0.5 - 0.001 for ϵ. As we expect, the smaller the value of ϵ is, the longer the running times of both algorithms are. APM is generally more efficient than AAM, particularly for smaller values of ϵ. This observation confirms our comparative analysis of the time complexities of these algorithms in Section 5. We set the value of ϵ to 0.1 for AAM and 0.001 for APM in our experiments to evaluate the improvement in effectiveness of answering queries achieved by the designs produced by AAM and APM for frequency-based cost, reported in Section 6.3, we set the value of ϵ to 0.1 and 0.001, respectively. According to Table 6, the running times of the algorithms for these values of ϵ are reasonable for a design-time task. As we have to run AAM 40 times per budget in the experiments using random costs, reported in Section 6.3, we set the value of ϵ to 0.3 in AAM for these experiments. Table 6 indicates that the running time of AAM with this value of ϵ is reasonable for a design time task.

Both APM and AAM use dynamic programming method and keep a table in the main memory to maintain the solutions of their subproblems. Table 7 shows the average memory usage of APM and AAM algorithms over domains M1, M2, and M3 using values from 0.5 - 0.001 for ϵ. Similar to running time, the smaller the value of ϵ is, the larger the memory AAM and APM need. Inter-

	ϵ	0.5	0.3	0.1	0.01	0.001
APM	Domain M1	0.204	0.205	0.204	0.204	0.204
	Domain M2	0.204	0.205	0.204	0.204	0.222
	Domain M3	0.184	0.183	0.182	0.184	0.184
AAM	Domain M1	0.209	0.209	0.209	-	-
	Domain M2	0.229	0.238	0.239	-	-
	Domain M3	0.188	0.188	0.188	-	-

Table 8: Average $p@3$ over all budgets for AAM and APM using different values of ϵ

estingly, AAM uses smaller amount of memory over domain M3 than M2 even though the size of M2 is smaller than M3. We have found that the distributions of costs and frequencies of concepts in domain M3 is more skewed than that of domain M2. Thus, the size of $\mathcal{C}_{rem}$ for domain M3 tends to be smaller than the one for domain M2. Hence, the amount of memory space required to construct the dynamic programming table for AAM in domain M3 is smaller than the one for M2. The size of the main memory table becomes very large (e.g for some budgets it exceeds the available main memory) for $\epsilon \leq 0.01$ in AAM and for $\epsilon \leq 0.001$ in APM. Our results in Section 6.3 indicates that one does not need such small values for ϵ, particularly for AAM, in order to find effective designs. Hence, in this paper we have not used such values for $epsilon$ for APM and AAM.

Scalability: One may have to set ϵ to values larger than 0.3 or 0.1 for AAM and 0.001 for APM in order to find the desired designs for large domains in reasonable amount of time and using modest memory overheads. Hence, we empirically examine the effect of changes on the values of ϵ on the effectiveness of the algorithms. Table 8 shows $p@3$ of APM and AAM algorithms over domain M2 and M3 using values between 0.5 and 0.001 for ϵ. The average values of $p@3$ delivered by the designs of AAM is relatively stable across different values of ϵ in both domains. Generally, the ranking qualities delivered by the designs of AAM tends to improve when using smaller value of ϵ. Table 8 shows that AAM with $\epsilon = 0.5$ provides better ranking qualities than APM with smaller values of $\epsilon = 0.5$, and a comparable ranking quality to AAM using $\epsilon = 0.1$. With this choice of ϵ, AAM requires significantly less amount of resources than the that of ideal value of ϵ or that of APM with $\epsilon = 0.001$ while sustaining its effectiveness. Except for some cases, e.g. $\epsilon = 0.5$ in domain M3, we observe almost a similar trend in the effect of the values for ϵ to in the effectiveness provided by the designs of APM across all domains.

7. CONCLUSION

Annotating the occurrences of entities in an unstructured or semi-structured text collection by their concepts improves the effectiveness of answering queries over the collection. Nonetheless, annotating the occurrences of a concept is resource intensive. Thus, an enterprise may have to select a subset of the concepts for annotation, called a *conceptual design*, whose cost of annotation does not exceed its budget and improves the effectiveness of answering queries the most. We formalized this problem, proved it to be NP-hard, and proposed two efficient approximation algorithms for it: Approximate Popularity Maximization (APM) and Approximate Annotation-benefit Maximization (AAM). Our empirical studies showed that APM and AAM efficiently compute conceptual designs and return effective designs.

8. ACKNOWLEDGMENTS

We thank Soravit Changpinyo, Alan Xia, and Naga Varun Dasari for their helps with data preparation and Wolfgang Nejdl and Elena Demidova for providing the query workload. The authors are supported by NSF grants CCF-0938071, CCF-0938064, and CNS-0716532. Arash Termehchy is also supported by a Yahoo! Key Scientific Challenges Award.

9. REFERENCES

[1] S. Abiteboul et al. *Web Data Management*. Cambridge University Press, 2011.

[2] M. Anderson et al. Brainwash: A Data System for Feature Engineering. In *CIDR*, 2013.

[3] P. N. Bennett, K. Svore, and S. T. Dumais. Classification-Enhanced Ranking. In *WWW*, 2007.

[4] B. Boehm, C. Abts, and S. Chulan. Software Development Cost Estimation Approaches, A Survey. *Annals of Software Engineering*, 10, 2000.

[5] S. Chakrabarti, K. Puniyani, and S. Das. Optimizing Scoring Functions and Indexes for Proximity Search in Type-annotated Corpora. In *WWW*, 2007.

[6] L. Chiticariu et al. Enterprise information extraction: recent developments and open challenges. In *SIGMOD*, 2010.

[7] J. Chu-Carroll et al. Semantic Search via XML Fragments: a High-Precision Approach to IR. In *SIGIR*, 2006.

[8] S. Dill et al. SemTag and Seeker: Bootstrapping the Semantic Web via Automated Semantic Annotation. In *WWW*, 2003.

[9] Elena Demidova and Xuan Zhou and Irina Oelze and Wolfgang Nejdl. Evaluating Evidences for Keyword Query Disambiguation in Entity Centric Database Search. In *DEXA*, 2010.

[10] H. GarciaMolina, J. Ullman, and J. Widom. *Database Systems: The Complete Book*. Prentice Hall, 2008.

[11] J. Graupmann, R. Schenkel, and G. Weikum. The SphereSearch Engine for Unified Ranked Retrieval of Heterogeneous XML and Web Documents. In *VLDB*, 2005.

[12] P. Gulhane et al. Web-Scale Information Extraction with Vertex. In *ICDE*, 2011.

[13] J. Huang and C. Yu. Prioritization of Domain-Specific Web Information Extraction. In *AAAI*, 2010.

[14] O. Ibarra and C. Kim. Fast Approximation Algorithms for The Knapsack and Sum of Subset Problems. *Journal of ACM*, 22, 1975.

[15] A. Jain, A. Doan, and L. Gravano. Optimizing SQL Queries over Text Databases. In *ICDE*, 2008.

[16] P. Kanani and A. McCallum. Selecting Actions for Resource-bounded Information Extraction using Reinforcement Learning. In *WSDM*, 2012.

[17] B. Korte and R. Schrader. On the Existence of Fast Approximation Schemes. *O.L. Mangasarian, R.R. Meyer, and S.M. Robinson (eds.) Nonlinear Programming, Academic Press, New York, 41 - 437*, 1981.

[18] P. Kungas and M. Dumas. Cost Effective Semantic Annotation of XML Schemas and Web Service Interfaces. In *SCC*, 2009.

[19] C. Manning, P. Raghavan, and H. Schutze. *An Introduction to Information Retrieval*. Cambridge University Press, 2008.

[20] A. McCallum. Information Extraction: Distilling Structured Data From Unstructured Text. *ACM Queue*, 2005.

[21] E. Riloff and R. Jones. Learning Dictionaries for Information Extraction by Multi-level Bootstrapping. In *AAAI*, 1999.

[22] M. Sanderson. Ambiguous Queries: Test Collections Need More Sense. In *SIGIR*, 2008.

[23] R. Schenkel, F. Suchanek, and G. Kasneci. YAWN: A Semantically Annotated Wikipedia XML Corpus. In *BTW*, 2007.

[24] W. Shen, P. DeRose, R. McCann, A. Doan, and R. Ramakrishnan. Toward Best-Effort Information Extraction. In *SIGMOD*, 2008.

[25] A. Termehchy, A. Vakilian, Y. Chodpathumwan, and M. Winslett. Cost Effective Conceptual Design for Semantic Annotation. Technical report, Oregon State University, 2013.