Exploiting Web Search to Generate Synonyms for Entities

Surajit Chaudhuri Venkatesh Ganti Dong Xin
Microsoft Research
Redmond, WA 98052
{surajitc, vganti, dongxin}@microsoft.com

ABSTRACT

Tasks recognizing named entities such as products, people names, or locations from documents have recently received significant attention in the literature. Many solutions to these tasks assume the existence of reference entity tables. An important challenge that needs to be addressed in the entity extraction task is that of ascertaining whether or not a candidate string approximately matches with a named entity in a given reference table. Prior approaches have relied on string-based similarity which only compare a candidate string and an entity it matches with. In this paper, we exploit web search engines in order to define new similarity functions. We then develop efficient techniques to facilitate approximate matching in the context of our proposed similarity functions. In an extensive experimental evaluation, we demonstrate the accuracy and efficiency of our techniques.

Categories and Subject Descriptors

H.2.8 [**Database Applications**]: Data Mining

General Terms

Algorithms

Keywords

Synonym Generation, Entity Extraction, Similarity Measure, Web Search

1. INTRODUCTION

Tasks relying on recognizing entities have recently received significant attention in the literature [10, 12, 2, 14, 11, 9]. Many solutions to these tasks assume the existence of extensive *reference entity tables*. For instance, extracting named entities such as products and locations from a reference entity table is important for several applications. A typical application is the *business analytics and reporting* system which analyzes user sentiment of products. The system periodically obtains a few review articles (*e.g.*, feeds from review website and online forums), and aggregates user reviews for a reference list of products (*e.g.*, products from certain manufacturers, or products in certain categories). Such a reporting application requires us to effectively identify mentions of those reference products in the review articles.

WWW 2009, April 20–24, 2009, Madrid, Spain.
ACM 978-1-60558-487-4/09/04.

Consider another application. The entity matching task identifies entity pairs, one from a reference entity table and the other from an external entity list, matching with each other. An example application is the *offer matching* system which consolidates offers (*e.g.*, listed price for products) from multiple retailers. This application needs to accurately match product names from various sources to those in the system's reference table, and to provide a unified view for each product.

At the core of the above two applications, the task is to check whether or not a *candidate string* (a sub-string from a review article or an entry from an offer list) matches with a member of a reference table. This problem is challenge because users often like to use phrases, which are not member of the reference table, to refer to some entities. These phrases can be an individual's preferred description of an entity, and the description is different from the entity's conventional name included in a reference table. For instance, consider a product entity "Lenovo ThinkPad X61 Notebook". In many reviews, users may just refer to "Lenovo ThinkPad X61 Notebook" by writing "Lenovo X61", or simply "X61". Exact match techniques, which insist that sub-strings in review articles match exactly with entity names in the reference table, drastically limit their applicability in our scenarios.

To characterize whether or not a candidate string matches with a reference entity string, an alternative approach is to compute the string similarity score between the candidate and the reference strings [10, 6]. For example, the (unweighted) Jaccard similarity[1] function comparing a candidate string "X61" and the entity "Lenovo ThinkPad X61 Notebook" would observe that one out of four distinct tokens (using a typical white space delimited tokenizer) are common between the two strings and thus measures similarity to be quite low at $\frac{1}{4}$. On the other hand, a candidate string "Lenovo ThinkPad Notebook" has three tokens which are shared with "Lenovo ThinkPad X61 Notebook", and thus the Jaccard similarity between them is $\frac{3}{4}$. However, from the common knowledge, we all know that "X61" does refer to "Lenovo ThinkPad X61 Notebook", and "Lenovo ThinkPad Notebook" does not because there are many models in the ThinkPad series. We observe a similar problem with other similarity functions as well. Therefore, the string-based similarity does not often reflect the "common knowledge" that users generally have for the candidate string in question.

[1]These similarity functions also use token weights, say IDF weights, which may in turn depend on token frequencies in a corpus or a reference table.

In this paper, we address the above limitation. We observe that the "common knowledge" is often incorporated in documents within which a candidate string is mentioned. For instance, the candidate string "X61" is very highly correlated with the tokens in the entity "Lenovo ThinkPad X61 Notebook". And, many documents which contain the tokens "X61" also mention within its vicinity the remaining tokens in the entity. This provides a stronger evidence that "X61" matches with "Lenovo ThinkPad X61 Notebook". In this paper, we observe that such "correlation" between a candidate string τ and an entity e is seen across multiple documents and exploit it. We propose new document-based similarity measures to quantify the similarity in the context of multiple documents containing τ. However, the challenge is that it is quite hard to obtain a large number of documents containing a string τ unless a large portion of the web is crawled and indexed as done by search engines. Most of us do not have access to such crawled document collections from the web. Therefore, we exploit a web search engine and identify a small set of very relevant documents (or even just their snippets returned by a web search engine) containing the given candidate string τ. We rely on these small set of highly relevant documents to measure the correlation between τ and the target entity e.

Note that our criteria matching τ and e needs the web search results for τ to obtain a highly relevant set of documents or snippets containing τ. Hence, evaluating the similarity between a candidate string with entities in a reference table in general may not be applicable. Our approach here is to first identify a set of "synonyms" for each entity in the reference table. Once such synonyms are identified, our task of approximately matching a candidate string with a reference entity is now reduced to match *exactly* with synonym or original entity names, i.e., the (sub)set of tokens in the candidate string is equal to the token set of either a synonym or of an original entity name. Methods which only support exact match between candidate strings and entities in a reference table are significantly faster (e.g., [3]).

In this paper, we focus on a class of synonyms where each synonym for an entity e is an *identifying* set of tokens, which when mentioned contiguously (or within a small window) refer to e with high probability. We refer to these *identifying token sets* as IDTokenSets. We only consider *IDTokenSets* for an entity e which consist of a subset of the tokens in e for two reasons. First, the reference entity tables are often provided by authoritative sources; hence, each entity name generally does contain the most important tokens required to identify an entity exactly but may also contain redundant tokens which are not required for identifying the entity. Therefore, it is sufficient to isolate the identifying subset of tokens for each entity as an *IDTokenSet*. The *IDTokenSets* of an entity can be considered as keys that uniquely refer to the original entity. Second, our target applications are mainly entity extraction from documents. These documents are mainly drawn from the web such as blogs, forums, reviews, queries, *etc*, where it is often observed that users like to represent a possibly long entity name by a subset of identifying tokens (*e.g.*, 1-3 keywords).

The main technical challenge in identifying *IDTokenSets* for an entity e is that the number of all token subsets of e could be fairly large. For example, the entity "Canon EOS Digital Rebel XTI SLR Camera" has 127 subsets. Directly evaluating whether or not each subset τ_e of e matches with e would require a web search query to be issued. Therefore, the main challenge is to reduce the number of web search queries issued to identify *IDTokenSets* for an entity. Our main insight in addressing this challenge is that for most entities, if the set $\tau_e \subset e$ of tokens identify an entity e then a set τ'_e where $\tau_e \subset \tau'_e \subset e$ also identifies e (*i.e.*, *subset-superset monotonicity*). In other words, if $\tau_e \subset \tau'_e \subset e$, then τ'_e is more correlated to e. This is reminiscent of the "apriori" property in the frequent itemset mining [1, 13], where a superset is frequent only if its subsets are frequent.

We assume that the *subset-superset monotonicity* is true in general and develop techniques which significantly reduce the number of web search queries issued. For example, suppose "Canon XTI" identifies "Canon EOS Digital Rebel XTI SLR Camera" uniquely. Hence we assume that any superset say "Canon EOS XTI" also identifies e_1 uniquely. Therefore, if we efficiently determine the "border" of *IDTokenSets* whose supersets are all *IDTokenSets* and whose subsets are not, then we can often significantly reduce the number of web search queries per entity. In this paper, we develop efficient techniques to determine the border efficiently. We further extend these techniques for multiple entities by taking advantage of entities which are structurally similar.

In summary, our contributions in this paper are as follows.

1. We consider a new class of similarity functions between candidate strings and reference entities. These similarity functions are more accurate than previous string-based similarity functions because they aggregate evidence from multiple documents, and exploit web search engines in order to measure similarity.

2. We develop efficient algorithms for generating *IDTokenSets* of entities in a reference table.

3. We thoroughly evaluate our techniques on real datasets and demonstrate their accuracy and efficiency.

The remainder of the paper is organized as follows. We define problem in Section 2. We develop several efficient algorithms for generating *IDTokenSets* in Section 3, and discuss some extensions in Section 4. We present a case study that uses *IDTokenSets* for entity extraction in Section 5. In Section 6, we discuss the experimental results. In Section 7, we review the related work. Finally, we conclude in Section 8.

2. PROBLEM DEFINITION

We first define the notation used in the paper. Let $\mathcal{E}$ denote the set of entities in a reference table. For each $e \in \mathcal{E}$, let $Tok(e)$ denote the set of tokens in e. For simplicity, we use e to denote $Tok(e)$. We use the notation τ_e to denote a subset of tokens of e. That is, $\tau_e \subseteq e$.

Recall that we focus on identifying token sets which are subsets of the token set of the entity. That is, an *IDTokenSet* of an entity e consists of a subset τ_e of tokens in e. In the following, we formally define *IDTokenSets*. As discussed earlier in Section 1, to characterize an *IDTokenSet*, we rely on a set of documents and analyze *correlations* between the candidate subset τ_e and the target entity e. If a subset τ_e identifies e, then a large fraction, say θ, of documents mentioning τ_e is likely to contain the remaining tokens in $e - \tau_e$. We first define the notion of a document *mentioning* a token subset.

DEFINITION 1. *Let d be a document and τ_e be a set of tokens. We say that d* mentions *τ_e if there exists a substring s of d such that $Tok(s) = \tau_e$.*

ID	Document
d_1	The All-New, 2009 Ford F150 takes on ...
d_2	Sony Vaio F150 is...business notebook...
d_3	An overview of the Ford F150 Pickup...

Table 1: A set of documents

EXAMPLE 1. *For instance, the document d_2 in Table 1 mentions the subset $\{Vaio, F150\}$, and the documents d_1 and d_3 mention the subset $\{Ford, F150\}$*

For each document that mentions a subset τ_e, we check whether the document also contains the remaining tokens in $e - \tau_e$. In the ideal case, a large fraction of these documents mention tokens in $e - \tau_e$ next to the mention of τ_e. However, this may be too constraining. Hence, we relax this notion in two ways. First, we want to parameterize the context window size p within which we expect to observe all tokens in $e - \tau_e$. Second, it may be good enough to find a significant fraction of tokens in $e - \tau_e$ within the context window of the mention of τ_e; the size of the fraction quantifies the evidence that τ_e refers to e.

DEFINITION 2. *(**p-window context**) Let $M = \{m\}$ be a set of mentions of τ_e in a document $d = t_1, \ldots, t_n$. For each mention m, let $c(m, p)$ be the sequence of tokens by including (at most) p tokens before and after m. The p-window context of τ_e in d is $C(\tau_e, d, p) = \bigcup_{m \in M} c(m, p)$.*

EXAMPLE 2. *For example, the 1-window context of "F150" in document d_2 of Table 1 is $\{$Vaio, F150, is$\}$ and that in d_1 and d_3 are $\{$Ford, F150, takes$\}$ and $\{Ford, F150, Pickup\}$, respectively.*

We now define the measure to quantify the evidence that τ_e refers to e in a document d. We first define the stricter notion of evidence g_1, where all tokens in $e - \tau_e$ are required to be present in the p-window context of τ_e.

$$g_1(\tau_e, e, d) = \begin{cases} 1 & if\ e \subseteq C(\tau_e, d, p) \\ 0 & otherwise \end{cases} \quad (1)$$

We now define a relaxed notion of evidence g_2 of a document referring to an entity e, which is quantified by the fraction of tokens in $e - \tau_e$ that are present in the p-window context of τ_e.

$$g_2(\tau_e, e, d) = \frac{\sum_{t \in C(\tau_e, d, p) \cap e} w(t)}{\sum_{t \in e} w(t)} \quad (2)$$

where $w(t)$ is the weight (*e.g.*, IDF weight [5]) of the token t.

The *IDTokenSets* problem is to generate for a given entity e all its *IDTokenSets* with respect to a document collection $\mathcal{D}$. In the ideal case, this set corresponds to a large collection of documents on the web which requires us to have access to a crawled repository of the web. Since this is hard to have access to in the scenarios we focus on, we exploit the web search engines to provide us a small set $W(\tau_e)$ of very relevant document snippets which are highly relevant for τ_e.

We are now ready to define the aggregated *correlation* between τ_e and e with respect to a document set $W(\tau_e)$. Informally, the aggregated correlation is the aggregated evidence that τ_e refers to e from all documents mentioning τ_e.

DEFINITION 3. **(Correlation)** *Given e, τ_e, a search engine W, we define the aggregated correlation $corr(\tau_e, e, W(\tau_e))$ as follows.*

$$corr(\tau_e, e, W(\tau_e)) = \frac{\sum_{d \in W(\tau_e), d\ mentions\ \tau_e} g(\tau_e, e, d)}{|\{d | d \in W(\tau_e), d\ mentions\ \tau_e\}|}$$

Given a correlation threshold θ, we say that τ_e is an *IDTokenSet* of e if $corr(\tau_e, e, W(\tau_e)) \geq \theta$.

EXAMPLE 3. *Let e ="Sony Vaio F150 Laptop" be the target entity, and $\tau_e = \{F150\}$ be the candidate subset. Suppose documents in Table 1 are obtained snippets from $W(\tau_e)$. Each document mentions $\{F150\}$. In order to validate whether $\tau_e = \{F150\}$ is an* IDTokenSet *of e, we compute:*
$g_1(\tau_e, e, d_1) = 0$, $g_1(\tau_e, e, d_2) = 1$, $g_1(\tau_e, e, d_3) = 0$.
Thus, $corr(\tau_e, e, W(\tau_e)) = \frac{1}{3} = 0.33$.
Suppose the token weights of $\{Sony, Vaio, F150, Laptop\}$ are $\{6.8, 9.5, 10.5, 6.5\}$. Using g_2, we have:
$g_2(\tau_e, e, d_1) = 0.29$, $g_2(\tau_e, e, d_2) = 0.80$, $g_2(\tau_e, e, d_3) = 0.29$.
Thus, $corr(\tau_e, e, W(\tau_e)) = \frac{1.38}{3} = 0.46$.

DEFINITION 4. *(**IDTokenSets Problem**) Given an entity e, a search engine W, and the correlation threshold θ, the* IDTokenSets *problem is to identify the set S_e of all subsets such that for each $\tau_e \in S_e$, $corr(\tau_e, e, W(\tau_e)) \geq \theta$.*

Using the above similarity function, adapting techniques which measure similarity between candidate strings and entities from reference tables directly is an expensive approach. Therefore, we pre-process the reference entity table and expand the original entities with their *IDTokenSets*. By generating accurate *IDTokenSets* off-line, we transform the *approximate match* against the reference entity table problem to an *exact match* over the set of *IDTokenSets*, thus significantly improving the efficiency and accuracy of the approximate lookup task.

3. GENERATING IDTOKENSETS

We now describe our techniques for efficiently generating *IDTokenSets* of a given set of entities. We first discuss the optimization criterion and the complexity of the optimal solution for generating *IDTokenSets* of a single entity. We then outline an algorithmic framework, under which, we develop two algorithms. We then extend these techniques to generate *IDTokenSets* for a set of entities, and take advantage of entities which are structurally similar. We show that one of the discussed algorithms is within a factor of the optimal solution.

3.1 Optimization Criterion

The input to our system is a set $\mathcal{E}$ of entities, and a search interface W. For each entity e, all subsets of e consist of the candidate space. For each subset τ_e of entity e, we want to validate whether τ_e is an *IDTokenSet* of e, using the measure in Definition 3. Specifically, the general framework to process an entity e is to validate each of its subsets τ_e, which consists of the following two steps:

1. Send τ_e as a query term to W, and retrieve $W(\tau_e)$ (title, URL and snippets) as the relevant documents;

2. Evaluate $corr(\tau_e, e, W(\tau_e))$, and report τ_e is an *IDTokenSet* if $corr(\tau_e, e, W(\tau_e)) \geq \theta$;

We consider the whole process to validate τ_e as an atomic operator, and notate it as $validate(\tau_e)$. Furthermore, we assume the cost of $validate(\tau_e)$ for different τ_e is roughly same since the most expensive part of $validate(\tau_e)$ is sending τ_e to W. Thus, in order to efficiently generate all *IDTokenSets* of an entity e, we need to reduce the number of web search queries we issued. The optimization is mainly based on the intuition that removing some tokens from a subset τ_e weakens the correlation between τ_e and e. On the other hand, adding more tokens (belong to e) to τ_e enhances the correlation between τ_e and e. This is formally characterized as the *subset-superset monotonicity* in Definition 5.

DEFINITION 5. *(**subset-superset monotonicity**) Given an entity e, let τ_e and τ'_e be two subsets of e, and $\tau_e \subset \tau'_e$. If τ_e is an* IDTokenSet *of e, then τ'_e is also an* IDTokenSet *of e.*

Based on the subset-superset monotonicity, if τ_e is an *IDTokenSet* of e, all subsets τ'_e ($\tau_e \subset \tau'_e \subseteq e$) are *IDTokenSets* of e, and thus can be pruned for validation. If τ_e is not an *IDTokenSet* of e, all subsets τ'_e ($\tau'_e \subset \tau_e$) are not *IDTokenSets* of e, and thus can be pruned for validation. Therefore, by appropriately schedule the order in which subsets τ_e are submitted for validation, we can reduce the number of web search queries. Before we present the detailed algorithms, we first discuss the optimal solution.

3.2 Complexity of the Optimal Algorithm

In order to exploit the subset-superset monotonicity, we use the lattice structure to model the partial order between all subsets. An example of subset-lattice of entity "Sony Vaio F150 Laptop" is shown in Figure 1.

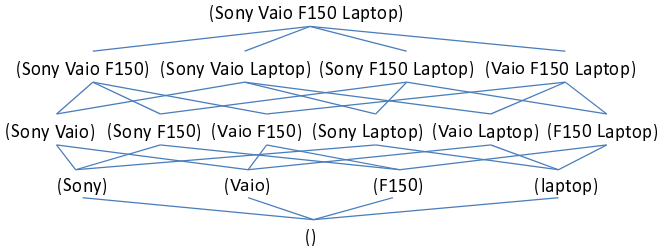

Figure 1: Subset-lattice of "Sony Vaio F150 Laptop"

The optimal algorithm is built upon the notion of *minimal positive subset* and *maximal negative subset*, as defined below.

DEFINITION 6. *Given an entity e, a subset τ_e is a minimal positive subset if $validate(\tau_e) = true$ and for all subsets $\tau'_e \subset \tau_e$, $validate(\tau'_e) = false$. Similarly, a subset τ_e is a maximal negative subset if $validate(\tau_e) = false$ and for all subsets τ'_e such that $\tau_e \subset \tau'_e \subseteq e$, $validate(\tau'_e) = true$.*

We use the notation $Cut(e)$ to denote the set of all minimal positive and maximal negative subsets. We now illustrate it with an example.

EXAMPLE 4. *Given the entity "Sony Vaio F150 Laptop", the subset $\tau_e^1 = \{sony, vaio, laptop\}$ is not an* IDTokenSet *since there are models other than F150 in the vaio series. Consequently, all subsets of $\{sony, vaio, laptop\}$ are not* IDTokenSets*. Because F150 is a popular ford truck, $\tau_e^2 = \{F150\}$ is not an* IDTokenSet *either. However, $\tau_e^3 = \{sony, F150\}$, $\tau_e^4 = \{vaio, F150\}$ and $\tau_e^5 = \{F150, laptop\}$ are all* IDTokenSets*. These five subsets constitute the cut. One can easily verify that all other subsets are either supersets of $\tau_e^3, \tau_e^4, \tau_e^5$ or subsets of τ_e^1, τ_e^2.*

Consider a special case where all subsets with $\frac{|e|}{2}$ tokens (suppose $|e|$ is even) are not *IDTokenSets* and all subsets with $\frac{|e|}{2}+1$ tokens are *IDTokenSets*. One can easily verify that all subsets with $\frac{|e|}{2}$ or $\frac{|e|}{2}+1$ tokens constitute $Cut(e)$, and $|Cut(e)|$ is exponential to $|e|$. For a given entity, a scheduling algorithm is optimal if it validates the minimal number of subsets. One can easily verify that any subset in the cut can not be pruned by other subsets, and thus has to be validated. The following lemma shows the connection between optimal solution and $Cut(e)$.

LEMMA 1. *Given an entity e, the optimal scheduling algorithm validates and only validates subsets in $Cut(e)$. In the worst case, the number of subsets in $Cut(e)$ is exponential to $|e|$.*

3.3 Algorithmic Framework

As shown in the previous subsection, given an entity e, it is sufficient to validate those subsets that are in $Cut(e)$. However, both the maximal negative and minimal positive subsets are not known beforehand. In this paper, we use a greedy algorithmic framework that iteratively probes a subset, validates it and prunes other subsets (if applicable). The algorithm stops when no subset is left undetermined.

The framework is outlined in Algorithm 1. Let $\mathcal{P}_e$ be the set of all subsets of e. Our task is to determine for all subsets in $\mathcal{P}_e$, whether they are *IDTokenSets* of e. The algorithm maintains a set of candidate subsets in $\mathcal{L}_e$. Initially, $\mathcal{L}_e = \mathcal{P}_e$. As soon as a subset $\tau_e \in \mathcal{L}_e$ is validated or pruned, τ_e is removed from $\mathcal{L}_e$. The algorithm repeats the following two steps until $\mathcal{L}_e = \phi$.

1. *validate-and-prune* (Line 4 to Line 9): It validates a subset τ_e. If τ_e is an *IDTokenSet*, all τ_e's supersets are determined to be *IDTokenSets*, and will be pruned from $\mathcal{L}_e$ for further validation. If τ_e is not an *IDTokenSet*, all τ_e's subsets are determined to be not *IDTokenSets*, and will be pruned from $\mathcal{L}_e$ as well.

2. *getnext* (Line 3): It determines which subset to visit next. We discuss various strategies for implementing *getnext* in this section.

3.4 Single-Entity Scheduling

We now describe two strategies to implement *getnext*. The depth-first scheduling starts with the maximal (or minimal) subset, and schedules subsets for validation by following the edges on the lattice structure. The max-benefit scheduling considers all subsets simultaneously: for each remaining subset, it computes the potential benefit for each subset, and picks the one with the maximal benefit.

Algorithm 1 Generating *IDTokenSets* for an Entity

Input: An entity: e, Search interface: W
the size of the context window: p,
number of top documents: k,
threshold for *IDTokenSet*: θ

1: Let $\mathcal{L}_e = \mathcal{P}_e$; //all subsets of e;
2: **while** ($\mathcal{L}_e$ is not empty)
3: $\tau_e = getnext(\mathcal{L}_e)$;
4: Submit τ_e to W, and retrieve $W(\tau_e)$;
5: **if** ($corr(\tau_e, e, W(\tau_e)) \geq \theta$) // τ_e is an *IDTokenSet*
6: Report τ_e and all its supersets as *IDTokenSets*;
7: Remove τ_e and all its supersets from $\mathcal{L}_e$;
8: **else**//τ_e is not an *IDTokenSet*
9: Remove τ_e and its subsets from $\mathcal{L}_e$;
10: **return**

3.4.1 Depth-first Scheduling

Given an entity e, all its subsets constitute a lattice (see Figure 1). The main idea of depth-first strategy is to start with a top root node (it can start at the bottom node as well) and recursively traverse the lattice structure. Suppose the algorithm reaches a node corresponding to a subset τ_e at some stage. The *getnext* step determines which subset to validate next. It consists of three steps:

1. Let τ_e^c be a child of τ_e. If $\exists \tau_e' \in \mathcal{L}_e$ such that $\tau_e' \subseteq \tau_e^c$ (note τ_e' could be τ_e^c itself), τ_e^c is a candidate for scheduling. That is, there is a descendent τ_e' whose status is unknown;

2. If step 1 did not find a candidate τ_e^c for scheduling, the algorithm looks for the siblings of τ_e. Let τ_e^s be a sibling of τ_e. If $\exists \tau_e' \in \mathcal{L}_e$ such that $\tau_e' \subseteq \tau_e^s$, τ_e^s is a candidate for scheduling;

3. If neither step 1 nor step 2 find a candidate for scheduling, the algorithm goes back to τ_e's parent τ_e^p, and restarts the step 1 on τ_e^p.

When multiple subsets (such as multiple children of τ_e or multiple siblings of τ_e) are available for scheduling, we rely on the intuition that the higher string similarity between the subset and e, the higher the possibility that this subset is an *IDTokenSet*. Since the depth-first scheduling starts from e, we expect to quickly find a subset that is not an *IDTokenSet* in order to prune all its descendent subsets. Therefore, we pick the candidate subset with the lowest string similarity (*e.g.*, the Jaccard similarity as defined in Section 2). Similarly, if the traversal starts from the bottom node, we will pick the candidate subset with the highest string similarity.

Theorem 1 gives a performance guarantee by the depth-first scheduling algorithm. The main insight is that using the depth-first scheduling, we will validate at most $|e| - 1$ subsets before we hit a subset belonging to the cut. We omit the detailed proof here.

THEOREM 1. *Given an entity e, let $DFS(e)$ be the number of validations (e.g., web search queries) performed by the depth-first scheduling, and let $OPT(e)$ be the number of validation performed by the optimal scheduling. We have $DFS(e) \leq |e| OPT(e)$.*

3.4.2 Max-Benefit Scheduling

Different from the depth-first scheduling, the max-benefit scheduling does not confine to the lattice structure. Instead, at any stage, all subsets in $\mathcal{L}_e$ are under consideration, and the one with the maximum estimated benefit will be picked.

The *getnext* step in the max-benefit scheduling works as follows. For each subset $\tau_e \in \mathcal{L}_e$, we can benefit in two ways from validating τ_e: $pos\_benefit(\tau_e)$ if $validate(\tau_e) = true$ or $neg\_benefit(\tau_e)$ if $validate(\tau_e) = false$. The benefit is simply computed by the number of subsets in $\mathcal{L}_e$ that are expected to be pruned. If $validate(\tau_e) = true$, the benefit of validating τ_e is defined as follows.

$$pos\_benefit(\tau_e) = |\{\tau_e' | \tau_e \subseteq \tau_e' \subseteq e \text{ and } \tau_e' \in \mathcal{L}_e\}| \quad (3)$$

If $validate(\tau_e) = false$, the benefit of validating τ_e is defined as follows.

$$neg\_benefit(\tau_e) = |\{\tau_e' | \tau_e' \subseteq \tau_e \text{ and } \tau_e' \in \mathcal{L}_e\}| \quad (4)$$

We consider three aggregate benefit formulation: max, min and avg, as defined as follows.

$$\begin{aligned} max(\tau_e) &= \max\{pos\_benefit(\tau_e), neg\_benefit(\tau_e)\} \\ min(\tau_e) &= \min\{pos\_benefit(\tau_e), neg\_benefit(\tau_e)\} \\ avg(\tau_e) &= \tfrac{1}{2}(pos\_benefit(\tau_e) + neg\_benefit(\tau_e)) \end{aligned}$$

Intuitively, max is an aggressive aggregate which always aims for the best; min is a conservative aggregate which guarantees for the worst scenario; and avg is in between the above two. For each of the aggregate option, the *getnext* step picks a τ_e with the maximum aggregated benefit.

3.5 Multiple-Entity Scheduling

In this subsection, we discuss the techniques for scheduling subset validation when the input consists of multiple entities, and our goal is to generate *IDTokenSets* for all entities. We first note that our techniques in this section improve the efficiency and the results are still correct. That is, the result would be the same as that of processing each entity independently, and taking the union of the results.

The intuition is as follows. Often, names of entities follow an implicit structure. The *IDTokenSets* of such structurally similar entities are also likely to follow the implicit structure. By exploring such structured information across entities, our scheduling strategy can be more efficient. Suppose there is a group of entities $e_1, e_2, \ldots, e_n$, which are structurally similar to each other. After the first i entities are processed, we may have a better idea on which subsets of e_{i+1} are *IDTokenSets* and which are not. For instance, both "lenovo thinkpad T41" and "lenovo thinkpad T60" belong to the thinkpad series from Lenovo. After processing "lenovo thinkpad T41", one may identify that $\{T41\}$ is an *IDTokenSet* of "lenovo thinkpad T41", and $\{T41\}$ belongs to the cut. By observing the structural similarity across entities, we may first validate $\{T60\}$ in its lattice structure. Depending on the outcome of validation, the scheduling algorithm may terminate early or proceed further.

In order to build the connection across multiple entities, we first group together entities that are structurally similar. For each group, we create a *group profile*, which aggregates statistics from entities in the group processed so far. Our new benefit estimation function for any subset exploits the the statistics on the group profile. We continue to apply

Algorithm 1 on each individual entity with a new *getnext* that leverages the group profile statistics.

3.5.1 Profile Grouping

Observe that our single entity scheduling algorithms operate on the subset lattice structure obtained by tokenizing an input entity. In order to share statistics for improved scheduling across entities, the statistics also have to be on the same subset lattice structure. Otherwise, it would be much harder to exploit them. Therefore, the main constraint on grouping multiple entities together for statistics collection is that we should be able to easily aggregate statistics across entity lattices.

In this paper, we take an approach of normalizing entity names based on "token level" regular expressions. That is, each of these normalization rules takes as input a single token and maps it to a more general class, all of which are accepted by the regular expression. The outcome is that entities which share the same normal form (characterized by a sequence of token level regular expressions) may all be grouped together. More importantly, they would share the same subset lattice structure. We further denote the normalized form shared by all entities in the group as *group profile*. Some example token level regular expressions are as follows.

- Regular expressions: $[0-9]+ \rightarrow PURE\_NUMBER$, $[A-Z][0-9]+ \rightarrow CHAR\_NUMBER$
- Synonyms: $\{red, green, blue, white, \ldots\} \rightarrow COLOR$, $\{standard, professional, enterprise\} \rightarrow VERSION$

Formally, we define the group and its profile as follows. An example is given in Example 5.

DEFINITION 7. *Let $e_1, e_2, \ldots, e_n$ be n entities. Let $\mathcal{N} = \{rule_1, rule_2, \ldots, rule_r\}$ be a set of single token normalization rules. We denote $\overline{e_i}$ as the normalized form of e_i after applying rules in $\mathcal{N}$. A set $\{e_1, e_2, \ldots, e_n\}$ form a group if $\overline{e_1} = \overline{e_2} = \ldots = \overline{e_n}$. The group profile is the normalized entity $\overline{e_i}$ $(i = 1, \ldots, n)$.*

EXAMPLE 5. *Suppose the normalization rule is $[A-Z][0-9]+ \rightarrow CHAR\_NUMBER$. Given two entities e_1="lenovo thinkpad T41" and e_2="lenovo thinkpad T60", their normalized forms are both $\overline{e_1} = \overline{e_2}$="lenovo thinkpad CHAR_NUMBER". Therefore, e_1 and e_2 form a group, and the group profile is "lenovo thinkpad CHAR_NUMBER".*

Based on the normalized rules, all input entities are partitioned into disjoint groups. Note that the normalized form of some entities may be the same as the original entity. This occurs when no normalization rules apply on the entity. Each entity where no normalization rules apply forms its own group and the multi-entity scheduling reduces to single-entity scheduling strategy.

3.5.2 Profile-Based Scheduling

After grouping entities into multiple partitions, we process entities one group at a time. In each group, we process entities one by one. Let $e_1, e_2, \ldots, e_n$ be entities in a group, and let e_p be the group profile. For any subset τ_{e_i} from e_i, there is a corresponding subset τ_{e_p} from e_p.

Assume the entities are processed in the order of $e_1, \ldots, e_n$. In the beginning, there are no statistics on e_p. We will use the single-entity scheduling algorithm (as shown in the previous subsection) to process e_1. Suppose the first i entities have been processed, and the next entity is e_{i+1}. The algorithm first updates the statistics on e_p using the validation results of e_i. For each subset τ_{e_p} in e_p, we keep two counters: $\tau_{e_p}.positive$ and $\tau_{e_p}.negative$. For each subset τ_{e_i} in e_i, if τ_{e_i} is an *IDTokenSet* of e_i, we increment $\tau_{e_p}.positive$ by 1. Otherwise, we increment $\tau_{e_p}.negative$ by 1. After e_p has accumulated statistics over a number of entities (*e.g.*, $i > 1$), we use the profile information to process e_{i+1}.

Similar to the max-benefit search, among all remaining subsets $\tau_{e_{i+1}}$ in $\mathcal{L}_{e_{i+1}}$, we will pick the one with the maximum benefit. The idea is that if a subset $\tau_{e_{i+1}}$ has higher probability to be an *IDTokenSet* of e_{i+1}, that is, $\tau_{e_p}.positive > \tau_{e_p}.negative$, we estimate its benefit using $pos\_benefit(\tau_{e_{i+1}})$ (*e.g.*, Equation 3). If $\tau_{e_{i+1}}$ has higher probability to be invalid, we estimate its benefit using $neg\_benefit(\tau_{e_{i+1}})$ (*e.g.*, Equation 4). If there is a tie between positive count and negative count, we do not consider $\tau_{e_{i+1}}$. If all subsets tie on the positive and negative counts, we switch to the single entity scheduling algorithm as we did for the first entity e_1. The benefit of a subset $\tau_{e_{i+1}}$ is formally defined as follows.

$$benefit(\tau_{e_{i+1}}) = \begin{cases} pos\_benefit(\tau_{e_{i+1}}) \\ \quad if\ \tau_{e_p}.postive > \tau_{e_p}.negative \\ neg\_benefit(\tau_{e_{i+1}}) \\ \quad if\ \tau_{e_p}.postive < \tau_{e_p}.negative \\ 0 \quad Otherwise \end{cases}$$

Observe that if for any entity e_{i+1}, where $(i > 0)$, if the profile e_p correctly accumulates the statistics such that for any $\tau_{e_{i+1}}$, $\tau_{e_{i+1}}$ is an *IDTokenSet* of e_{i+1} iff $\tau_{e_p}.positive > \tau_{e_p}.negative$, then the profile-based scheduling algorithm for e_{i+1} is optimal. That is, we directly validate the subsets on the $Cut(e_{i+1})$, thus mimicking the optimal algorithm would. In our experiments, we observe that the simple benefit function as defined above works well. Our algorithmic framework is able to take other benefit functions, and we intend to further explore them in the future.

4. EXTENSIONS

In this section, we discuss two extensions to our approach. In the first extension, we show how to incorporate additional constraints to prune candidate subsets for generating *IDTokenSet*. Such constraints are especially meaningful for entities with a large number of tokens. In the second extension, we relax the definition of mention (Definition 1) to further enrich the applicability of our techniques.

4.1 Constrained IDTokenSet Generation

In the above section, we assume all subsets are candidates for generating *IDTokenSet*. In some scenarios, users may have additional constraints that can be applied to prune some subset candidates. It is especially beneficial if the constraint can be evaluated before the subset is validated. Thus, we can save the cost of validation. To incorporate constraints into Algorithm 1, we simply replace $\mathcal{L}_e$ (the candidate space) by the set of candidates which satisfy the constraints. The rest of the algorithm remains the same.

4.2 Gap Mentions

In Example 3, $\{Sony, F150\}$ is an *IDTokenSet* of "Sony Vaio F150 Laptop". However, a document may mention

"Sony PCG F150" instead of "Sony F150". In Definition 1, we require that all tokens in a mention be contiguous within a document. Therefore, we would not identify the mention "Sony PCG F150". We now relax the definition of document mentioning a candidate by relaxing the requirement that a document sub-string consist of a contiguous set of tokens in a document. Instead, we may consider all sets of tokens which are "close" to each other within a window. Informally, we consider w-gap token sets from the document where the maximum gap (*i.e.*, number of intervening tokens) between neighboring tokens in the subset is no more than w. A w-gap token set that exactly matches with an *IDTokenSet* is called *w-gap mention*. w controls the proximity of the tokens, $w = 0$ means that the token set is a contiguous token set in the document d, $w = \infty$ means any subsets of tokens in d may be a valid mention. Typically, one may set $w = 0$ for longer documents such as web pages, and set $w = \infty$ for short documents such as queries, titles or snippets.

5. CASE STUDY: ENTITY EXTRACTION

Here we describe a case study that uses *IDTokenSets* to facilitate fast and accurate entity extraction. We assume the extraction task is based on a reference entity table, as demonstrated by the example applications in Section 1. The system architecture, which is outlined in Figure 2, consists of two phases: the offline phase and the online phase.

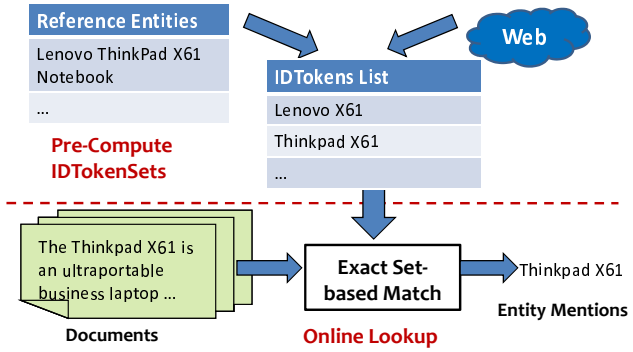

Figure 2: System Framework for Entity Extraction

The offline phase generates the *IDTokenSets* for each reference entity and constructs an exact lookup structure over the *IDTokenSets*. The online phase extracts sub-strings as candidates from query documents and checks them against the lookup structure for exact match. We observe that users often use different token order in mentioning an entity. For instance, both "sony F150 notebook" and "sony notebook F150" refer to the same entity. Therefore, we apply the set-based exact match criterion (or, the Jaccard similarity with threshold 1) between candidates from query documents and *IDTokenSets*. In doing this, we re-order tokens in each *IDTokenSet* according to a global order (say, lexicographic). The tokens in the candidates are also re-ordered according to the same order.

In order to efficiently extract candidates from query documents, we apply the optimization techniques used in [6]. First, we create a token table which keeps all distinct tokens appearing in the generated *IDTokenSets*. Given a query document, we first check each token against the token table, and only keep those *hit-tokens* (*i.e.*, tokens appearing in the table). We denote a set of contiguous *hit-tokens* as *hit-sequence*. Suppose we derive h *hit-sequences* from a query document. The second optimization exploits the concept of *strong-token*. From each *IDTokenSet*, we identify the token with the least frequency over the corpus. For instance, from the *IDTokenSet* $\{sony, F150, notebook\}$, one may extract $F150$ as the *strong-token*. Since we enforce exact match, a *strong-token* has to be matched between any candidate and its matched *IDTokenSet*. We put *strong-tokens* from all *IDTokenSets* into a *strong-token* table. For each *hit-sequence* derived from the first step, we check whether it contains a strong token. A *hit-sequence* is pruned immediately if it does not contain a *strong-token*. For all the remaining *hit-sequences*, we will enumerate sub-strings with length up to L (suppose the longest *IDTokenSet* has L tokens), while ensuring that each of which contains at least one *strong-token*. We then lookup each of these sub-strings against the lookup structure.

6. PERFORMANCE STUDY

We now present the results of an extensive empirical study to evaluate the techniques described in this paper. We use real data sets for the experiments. The reference entity table is a collection of $200k$ product names (*e.g.*, consumer and electronics, bicycles, shoes, *etc*). The number of tokens in the product names varies from 2 to 10, with 3.6 tokens on average.

The major findings of our study can be summarized as follows:

1. **High quality *IDTokenSets***: the document-based measure performs significantly better than the traditional string-based similarity measure in determining *IDTokenSets*;
2. **Manageable size**: the number of *IDTokenSets* that we generated is within reasonable size (*i.e.*, generally 2-4 times of the original entity number);
3. **Efficient generation**: our proposed algorithms are able to prune more than 80% of the web queries in generating *IDTokenSet*;
4. **Fast entity extraction**: the case study on entity extraction shows that using *IDTokenSets*, our system is able to process 200 documents per second, where each document contains $3,689$ tokens on average;

In the remaining of this section, we show the experimental results in terms of quality of *IDTokenSets*, cost of materializing *IDTokenSets* and the number of *IDTokenSets*. We also report the performance of the entity extraction application. We use Live Search API [2] to retrieve web documents.

6.1 Quality of IDTokenSets

To examine the quality of the *IDTokenSets*, we compare our proposed document-based measures with the traditional string-based similarity measure (*e.g.*, weighted Jaccard similarity). We use Live Search to retrieve top-10 results. Since we only consider title, url and snippets, which are succinct in general, we set $w = \infty$ in determining w-gap mentions (Section 4.2), and $p = \infty$ in determining p-window context (Definition 2). We notate **Corr1** for the document-based measure with g_1 (Equation 1), **Corr2** for the document-based

[2]http://dev.live.com/livesearch/

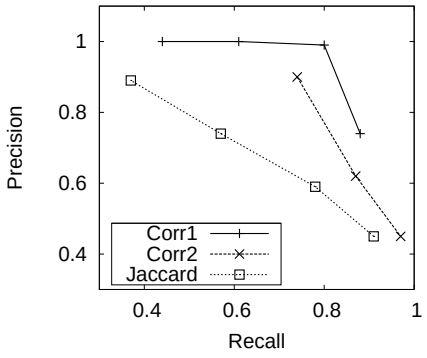

Figure 3: Precision-Recall on Bicycle Synonyms

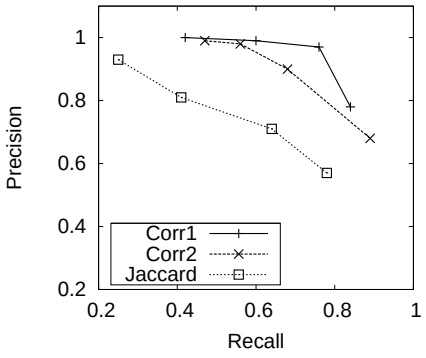

Figure 4: Precision-Recall on Laptop Synonyms

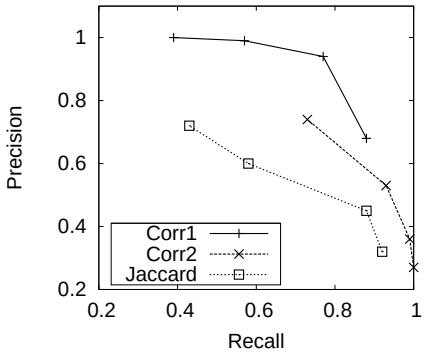

Figure 5: Precision-Recall on Shoe Synonyms

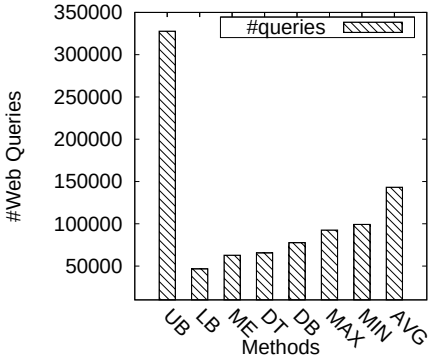

Figure 6: Generating *IDTokenSet* for $10k$ Product Names

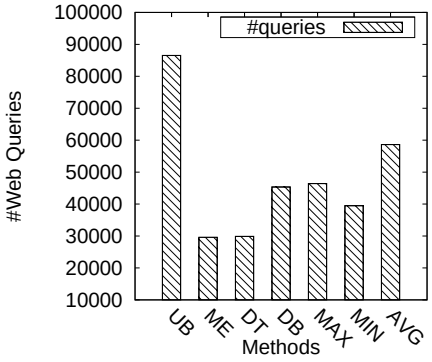

Figure 7: Generating *IDTokenSet* for $10k$ Product Names

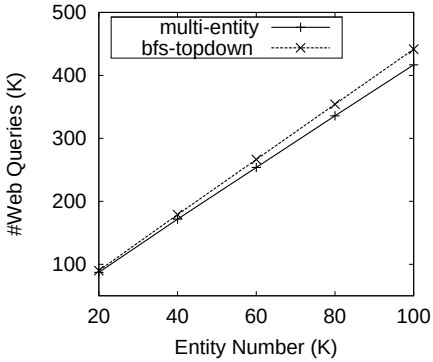

Figure 8: Number of Web Queries w.r.t. Entity Number

measure with g_2 (Equation 2), and **Jaccard** for the string-based weighted Jaccard Similarity. The experiments are conducted on three difference categories: bicycles, shoes and laptops. In each category, we sample 100 product names, and manually label all subsets which are *IDTokenSets*. By varying the threshold in (0.3, 0.5, 0.7, 0.9) for *Corr1*, (0.6, 0.7, 0.8, 0.9) for both *Corr2* and *Jaccard*, we plot the precision-recall curves in Figures 3-5.

We observe that in all three categories, the document-based measures are significantly better than the string-based measure. Among two variants of document-based measures, *Corr1* performs better than *Corr2*. We notice that *Corr1* is a fairly strict measure since it requires all tokens (in the original entity) to be presenting in at least 50% of the documents (assuming threshold is 0.5). When the threshold is 0.5, it achieves 80% recall and more than 97% precision.

The recall drops when we increase the threshold. This is expected since not all documents mention all tokens in the context. To improve the recall, we found it is important to recognize token-synonyms in matching tokens. For instance, the token in the product title may be "bicycle", and in the document, users may write "bike". Given the prior knowledge that we are processing bicycle names, "bike" can be matched to "bicycle". In this experiment, we only use the exact token matching, and the preliminary results is already very promising. We leave the robust token matching as future work.

6.2 Cost of Materializing IDTokenSets

Here we compare the computational performance using various strategies for generating *IDTokenSets*, as discussed in Section 3. The computation time is roughly proportional to the number of web queries. Since the time needed for a web query relies on network traffic condition and server loading status, we compare the number of web queries instead.

We use *Corr1* as the measure, and fix the threshold to 0.5. The reference entity table consists of $10k$ electronic and consumer product names. Figure 6 shows the number of web queries used by the following methods: *dfs-topdown* (**DT**), the depth-first scheduling starting from e for all entities e in the reference table; *dfs-bottomup* (**DB**), the depth-first scheduling starting from ϕ for all e; *max-max* (**MAX**), the max-benefit scheduling using *max* benefit function; *max-min* (**MIN**): the max-benefit scheduling using *min* benefit function; *max-avg* (**AVG**), the max-benefit scheduling using *avg* benefit function; *multi-entity* (**ME**): the multiple entity scheduling algorithm; *upper-bound* (**UB**), the total number of subsets; and *lower-bound* (**LB**), the total number of subsets in the cut (defined in Section 3.2).

We observe that *multi-entity* performs the best. It is close to the optimal scheduling in that the number of web queries is only 1.34 times of that in the lower bound. Comparing to the upper bound, it prunes more than 80% of web queries. Within the single entity scheduling, the depth-first

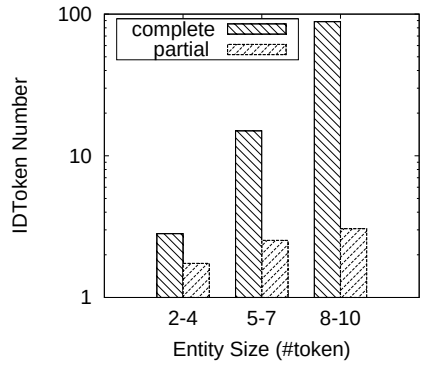

Figure 9: Number of *IDTokenSet* w.r.t. Entity Size

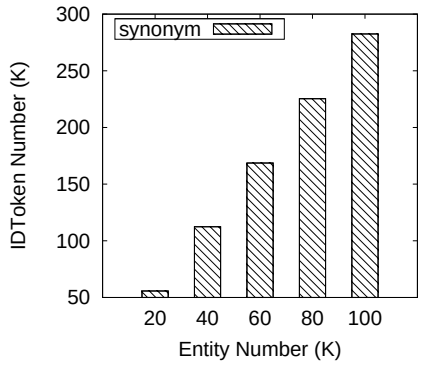

Figure 10: Number of *IDTokenSet* w.r.t. Entity Number

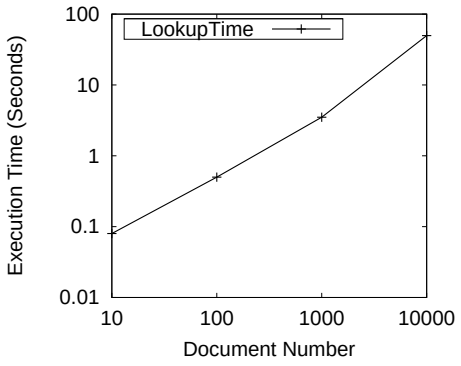

Figure 11: Query Performance

scheduling performs better than the max-benefit scheduling. Within the depth-first scheduling, the top-down scheduling strategy is better than the bottom-up scheduling strategy. This matches the intuition that *IDTokenSets* generally share more tokens with the original entities.

We also impose an additional constraint that each subset has to appear as a contiguous substring in some documents (as we discussed in Section 4.1). In order to evaluate the constraint, we still leverage the web search engine to retrieve the relevant documents. However, sending each subset to the web search engine is equally expensive as validating the subset. Instead, we using the following heuristic to evaluate the constraint. For each entity e, we will first send e as a query to the web search engine, and retrieve top-K documents (*i.e.*, title, url and snippets), where K could be a large number. We then evaluate the constraint by checking whether a subsets of e appears contiguously in at least one of these documents.

Figure 7 shows the performance by different methods, which is similar to that in the complete lattice case. Notice that different from the complete lattice case where we can compute the lower-bound of the web API calls, we are not able to obtain the lower-bound in the constrained case where candidate subsets form a partial lattice structure. This is because the optimal solution may pick any subsets in the complete lattice in order to prune subsets in the partial lattice. The optimal solution is essentially a set covering problem, which is NP-hard.

The last experiment in this subsection is to show the scalability of the algorithms. We fix a mixed strategy such that for all entities with no more than 5 tokens, we apply the complete lattice model; and for entities with more than 5 tokens, we enforce the same constraint in Figure 7. Figure 8 shows the number of web queries by *multi-entity* and *dfs-topdown*, with respect to different number of input entities. We observe that both methods are linearly scalable to the number of input entities. The performance gain of *multi-entity* is not significant in our experiment. This is because our entity database is rather diversified. However, it shows a clear trend that *multi-entity* benefits more by increasing the entity number. By increasing the number of entities, the grouping method is able to identify more entities which have similar structure to each other. Hence, the *multi-entity* has better chance to locate subsets in the Cut in scheduling.

6.3 Number of IDTokenSets

We pre-compute the *IDTokenSet* to support efficient approximate match on-the-fly. It is important to show that the *IDTokenSets* are within the manageable size. Figure 9 reports the average number of *IDTokenSets* per entity with respect to different entity size (*e.g.*, number of tokens in the entity). In each size group, we compute the number of *IDTokenSets* for both the complete lattice case and partial lattice case (using the same constraint in Figure 7). We observe that the constraint is quite effective in reducing the number of *IDTokenSets*. Actually, the constraint is also meaningful since not all *IDTokenSets* are conventionally used by users. Figure 10 shows the number of *IDTokenSets* by increasing the number of entities, using the same experimental configuration in Figure 8. In general, the number of *IDTokenSets* is about 2-4 times of the entity number.

6.4 Performance on Entity Extraction

In the last experiment, we briefly show the performance on entity extraction, using the generated *IDTokenSets*. The input string is a collection of $10,000$ documents. On average, each document contains $3,689$ tokens. The reference entity table includes $200k$ entities, from which, we generate $592k$ *IDTokenSets*. We extract all substrings from documents with length up to 10, and apply set based exact match (using a hash-table) over the *IDTokenSets*. We use the filter ideas discussed in Section 5 to prune non-interested substrings. The experiments are conducted on a 2.4GHz Intel Core 2 Duo PC with 4GB RAM, and the execution time is reported in Figure 11. Our system is fairly efficient in that it is able to process around 200 documents per second.

7. RELATED WORK

A number of techniques have been proposed for using dictionary information in entity extraction. Cohen and Sarawagi [10] exploited external dictionaries to improve the accuracy of named entity recognition. Agrawal *et al.* [2] introduced the "ad-hoc" entity extraction task where entities of interest are constrained to be from a list of entities that is specific to the task, and have also considered approximate match based on similarity functions.

Approximate-match based dictionary lookup was studied under the context of string similarity search in application scenarios such as data cleaning and entity extraction

(e.g., [7, 8, 4]). All these techniques rely on similarity functions which only use information from the input string and the target entity it is supposed to match. In contrast, we develop a new similarity scheme which exploits evidence from a collection of documents.

Our work is related to synonym detection, which is the problem of identifying when different references (i.e., sets of attribute values) in a dataset correspond to the same real entity. Synonym detection has received significant attention in the literature [14]. Most previous literature assumed references having a fair number of attributes. While additional attributes or linkage structures may help provide extra evidence, in this paper, we assume we are only given the list of entity names which is usually the case.

An alternative approach for generating *IDTokenSets* could be to segment the original entities into attributes. Consider the entity "Lenovo ThinkPad X61 Notebook", where the tokens can be tagged as follows: *Lenovo* (Brand Name), *ThinkPad* (Product Line), *X61* (Product Model) and *Notebook* (Product Type). A rule based approach then may suggest that *Product Model* is specific enough to refer to a product, and hence "X61" is a valid *IDTokenSet*. This rule based approach has two limitations. First, not all entities have a clear schema for segmentation. For instance, one may not be able to segment a movie name. Second, even for some entities that can be segmented, the rule based approach is not robust. Consider another product entity "Sony Vaio F150 Laptop", *F150* will be tagged as product model. Hence the rule based approach may conclude that "F150" is an *IDTokenSet* of "Sony Vaio F150 Laptop". While actually, from the common knowledge, "F150" is better known as a Ford vehicle. In contrast, our techniques do not assume that each entity is segmentable into attribute values, and we do not assume the availability of a robust segmentation technique.

Turney [15] introduced a simple unsupervised learning algorithm that exploits web documents for recognizing synonyms. Given a problem word and a set of alternative words, the task is to choose the member from the set of alternative words that is most similar in meaning to the problem word. In this paper, we focus on efficiently generating *IDTokenSets* for a large collection of entities. The search space is significantly larger than that in [15], and we focus on methods that minimize the generation cost.

The subset-superset monotonicity exploited in this paper is related to the "apriori" property used in many frequent itemset mining algorithms [1, 13]. The difference is that the subset-superset monotonicity prunes computation in two directions. That is, if τ is a valid *IDTokenSet*, all τ's supersets are pruned for validation; if τ is not a valid *IDTokenSet*, all τ's subsets are pruned for validation. While in frequent itemset mining, the "apriori" property only prunes computation in one direction (*e.g.*, if an itemset τ is not frequent, only its supersets are pruned).

8. DISCUSSION AND CONCLUSIONS

In order to support fast and accurate approximate entity match, we propose a new solution by exploiting *IDTokenSet*. Our approach differs from many previous methods in two folders: first, we pre-compute a list of *IDTokenSets* for each entity. Specifically, we use the document-based similarity measure to validate *IDTokenSets*, and leverage web search to retrieve related documents; and Second, we apply exact set-based match over the *IDTokenSets* at matching phase. This paper mainly focuses on the quality of the *IDTokenSets*, as well as efficient algorithms in generating *IDTokenSets*. We show the document-based measure is significantly better than the traditional string-based similarity measure. Several algorithms are proposed to efficiently generate *IDTokenSets* by pruning majority number of web queries.

We plan to explore several directions in future work. First, we studied a document-based similarity measure in this paper. One extension is to build a classifier with features derived from multiple documents. Second, we will consider an alternative computation model where the evidence documents are available for direct access (*e.g.*, scan all documents). Specifically, we will exploit batched processing by simultaneously generating *IDTokenSets* for all entities.

9. REFERENCES

[1] R. Agrawal, T. Imielinski, and A. Swami. Mining association rules between sets of items in large databases. In *SIGMOD Conference*, pages 207–216, 1993.

[2] S. Agrawal, K. Chakrabarti, S. Chaudhuri, and V. Ganti. Scalable ad-hoc entity extraction from text collections. In *VLDB*, 2008.

[3] A. V. Aho and M. J. Corasick. Efficient string matching: An aid to bibliographic search. *Commun. ACM*, 18(6):333–340, 1975.

[4] A. Arasu, V. Ganti, and R. Kaushik. Efficient exact set-similarity joins. In *VLDB*, pages 918–929, 2006.

[5] R. A. Baeza-Yates and B. A. Ribeiro-Neto. *Modern Information Retrieval*. ACM Press/Addison-Wesley, 1999.

[6] K. Chakrabarti, S. Chaudhuri, V. Ganti, and D. Xin. An efficient filter for approximate membership checking. In *SIGMOD Conference*, pages 805–818, 2008.

[7] A. Chandel, P. C. Nagesh, and S. Sarawagi. Efficient batch top-k search for dictionary-based entity recognition. In *ICDE*, page 28, 2006.

[8] S. Chaudhuri, V. Ganti, and R. Kaushik. A primitive operator for similarity joins in data cleaning. In *ICDE*, page 5, 2006.

[9] T. Cheng, X. Yan, and K. C.-C. Chang. Entityrank: Searching entities directly and holistically. In *VLDB*, pages 387–398, 2007.

[10] W. W. Cohen and S. Sarawagi. Exploiting dictionaries in named entity extraction: combining semi-markov extraction processes and data integration methods. In *KDD*, pages 89–98, 2004.

[11] X. Dong, A. Y. Halevy, and J. Madhavan. Reference reconciliation in complex information spaces. In *SIGMOD Conference*, pages 85–96, 2005.

[12] V. Ganti, A. C. König, and R. Vernica. Entity categorization over large document collections. In *KDD*, pages 274–282, 2008.

[13] J. Han and M. Kamber. *Data Mining: Concepts and Techniques*. Morgan Kaufmann, 2001.

[14] N. Koudas, S. Sarawagi, and D. Srivastava. Record linkage: similarity measures and algorithms. In *SIGMOD Conference*, pages 802–803, 2006.

[15] P. D. Turney. Mining the web for synonyms: Pmi-ir versus lsa on toefl. *CoRR*, cs.LG/0212033, 2002.