

A Framework for Robust Discovery of Entity Synonyms

Kaushik Chakrabarti, Surajit Chaudhuri, Tao Cheng, Dong Xin *

Microsoft Research
One Microsoft Way
Redmond, WA 98052
{kaushik, surajitc, taocheng}@microsoft.com, dongxin@google.com

ABSTRACT

Entity synonyms are critical for many applications like information retrieval and named entity recognition in documents. The current trend is to automatically discover entity synonyms using statistical techniques on web data. Prior techniques suffer from several limitations like click log sparsity and inability to distinguish between entities of different concept classes. In this paper, we propose a general framework for robustly discovering entity synonym with two novel similarity functions that overcome the limitations of prior techniques. We develop efficient and scalable techniques leveraging the MapReduce framework to discover synonyms at large scale. To handle long entity names with extraneous tokens, we propose techniques to effectively map long entity names to short queries in query log. Our experiments on real data from different entity domains demonstrate the superior quality of our synonyms as well as the efficiency of our algorithms. The entity synonyms produced by our system is in production in Bing Shopping and Video search, with experiments showing the significance it brings in improving search experience.

Categories and Subject Descriptors

H.2.8 [DATABASE MANAGEMENT]: Database applications—*Data mining*; H.3.3 [INFORMATION STORAGE AND RETRIEVAL]: Information Search and Retrieval

Keywords

entity synonym, robust synonym discovery, pseudo document similarity, query context similarity

1. INTRODUCTION

People often use several alternate strings to refer to the same named entity. For example, the product “Canon EOS 400d Digital Camera” is also referred to as “canon rebel xti” and “canon kiss k”, the movie “Harry Potter and the Half Blood Prince” is also referred to as “harry potter 6” or simply “half blood prince”, “Washington State Department of Licensing” is also referred to as “wa dol” and

*Work done while author at MSR. Author’s current address: Google.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

KDD’12, August 12–16, 2012, Beijing, China.

Copyright 2012 ACM 978-1-4503-1462-6 /12/08 ...\$15.00.

“wa dmv”. We refer to these alternate strings as *entity synonyms*. Knowing the synonyms for the entities of interest is critical in effective search over entities.

At Microsoft, two concrete examples of searching over entities are Bing Shopping search and Video search (the two target applications of this work). In shopping search, the engine will fail to return the product entity named “Canon EOS 400d Digital Camera” for query “canon rebel xti” if the product description does not mention “rebel xti” and it does not know they are synonyms. Similarly in video search, a user asking for “harry potter 6” will not be able to find the “Harry Potter and the Half Blood Prince” video if the engine does not know they are synonymous. Typically, such vertical search applications support search over a catalog of entities (e.g., product entities, video entities). This work aims at discovering the synonyms for entities based on entity names (e.g., product names, video titles) offline and enriching the catalogs with the discovered synonyms. Synonym-enriched catalog can help boost recall and improve precision, and therefore help improve users’ search experience. We will concretely study the positive effect of synonyms in improving these two search verticals in Section 7.

Entity synonyms are also very useful in many other applications. Applications like Voice of the Customer (VoC) and Social Media Analytics need to recognize mentions of named entities (e.g., products, organizations, people) in text documents (e.g., web pages, blogs, forum posts, tweets). Typically, these applications maintain a *reference table of entities* and require identification of mentions of *only* these entities in the documents [1, 3, 6]. For example, in VoC, an enterprise is typically interested in mining customer sentiment of its own and its competitors’ products, so it needs to identify *only* those products in web documents. Exact matching and even approximate matching techniques, based on string similarity functions (e.g., Jaccard similarity), will miss many mentions. For example, “canon rebel xti” has very low string similarity with “Canon EOS 400d Digital Camera”.

Review of State-of-the-art: To address the above needs, several similarity functions to automatically discover synonyms for entities have been proposed:

Click Similarity: Cheng et. al. identifies synonyms of entities using query click logs of a web search engine [6, 7]. They first identify webpage urls that are “good representatives” of the entity; for example, the urls clicked by users when the reference entity string is issued as query can be considered as good representatives. They then identify queries that have clicked at least one of the urls: these are “candidate synonyms strings” of the entity (henceforth, simply referred to as candidate synonyms). Finally, they identify the candidates that are strongly and exclusively related to the entity by inspecting click patterns on those representative urls; this is captured using two similarity functions which we refer to as *click similarity*

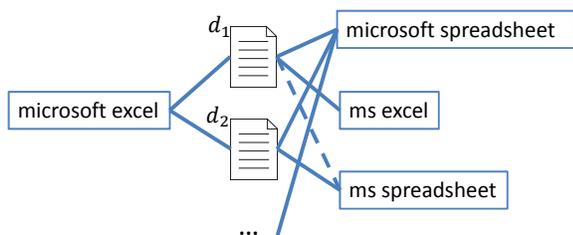


Figure 1: Motivating Example

(ClickSim in short).

Document Similarity: Turney proposes to identify synonyms based on cooccurrence in web documents [18]. If the set of documents where the entity reference string occurs is strongly correlated with the set where a candidate synonym string occurs, it is adjudged a synonym of the entity. This is captured using a similarity measure which we refer to as *document similarity* (DocSim in short).

Distributional Similarity: The distribution hypothesis is that similar terms appear in similar contexts [11]. Although this has not been applied to entities, it has been used to compute semantically related terms (based on similarity of the contexts of their mentions in documents) [17] (DistSim in short).

Limitations of State-of-the-art: The above similarity measures suffer from several limitations:

- **Click Log Sparsity:** Often, many true synonyms of an entity are tail queries, i.e, they are asked by very few users and thus there are few clicked documents; this is often referred to as the *click log sparsity* problem. They often have few or no clicked documents in common with the representative documents for the entity. ClickSim will miss these synonyms.

We illustrate this using a real example from our experiments as shown in Figure 1. “microsoft excel” is the reference entity string, and “microsoft spreadsheet”, “ms excel” and “ms spreadsheet” are three candidate synonyms. A solid edge between a query and a document represent a click on the document for that query. d_1 and d_2 are the representative documents for “microsoft excel”. “ms spreadsheet” is a tail query and has only one clicked document in common (d_2) with the representative documents for “microsoft excel”. Assuming the threshold is 2, “ms spreadsheet”, in spite of being a true synonym, will not be adjudged a synonym.¹ ClickSim often misses such tail queries which results in *poor recall*. Note that this can be mitigated by lowering the threshold but this leads to drop in precision.

- **Inability to distinguish entities of different classes:** Consider the entity “microsoft excel” and the candidate synonym “ms excel tutorial”. They might share many clicks with each other, hence it might be adjudged a synonym by ClickSim. Similarly, they might cooccur in many documents, hence it might be adjudged a synonym by DocSim as well. They are indeed very related but they do not belong to the same concept class; one is a tutorial whereas the other is a software product. To be a synonym, the entities must not only be highly related but also belong to the same concept class; the above techniques do not attempt to enforce the latter constraint. This leads to drop in precision.

Our Contributions: Our main contributions are summarized as follows:

- **General framework:** We first propose a set of *simple and natural properties* that entity synonyms should satisfy. We develop a novel synonym discovery framework that takes the individual sim-

¹For simplicity, we are thresholding based on the intersection count in this example. In practice, click ratios are used. The above limitation applies in that case as well.

ilarity values as input and combines them and produces synonyms that satisfy the above properties. Our synonym framework differs from a standard classification model with the similarity values as features as the former enforces the synonym properties while the latter does not (Section 2).

- **Novel similarity measures:** We propose two novel similarity measures, *Pseudo Document Similarity* (PseudoDocSim in short) and *Query Context Similarity* (QCSim in short), to identify synonyms that overcome the limitations of ClickSim and DocSim. One of the main technical challenges to overcome the click log sparsity is whether we can derive additional “edges” between documents and queries in the click graph. We address this challenge using the following insight: even if a (tail) query q does not click on a document d , we can infer whether d is related to q by observing other queries that clicked on d . For example, in Figure 1, “ms spreadsheet” did not click on d_1 but “microsoft spreadsheet” and “ms excel” did. There is enough evidence that d_1 is related to “ms spreadsheet”, so we add a new edge between “ms spreadsheet” and d_1 (shown by the dotted edge in Figure 1). In the above example, even with threshold 2, we will judge “ms spreadsheet” as a synonym. We propose the PseudoDocSim function that leverages the new edges to achieve higher recall than ClickSim without drop in precision (Section 4).

- **Efficient and scalable algorithms:** Often, we need to generate synonyms for millions of entities. To compute PseudoDocSim, we need to check the set containment of the tokens in each candidate synonym string and the “pseudo document” (bag of words constructed by concatenating the queries that clicked on it) of each representative document of the entity. This “all pairs” containment checking per entity is computationally challenging. We address this challenge using the following insight: for each entity, there is a high degree of overlap of tokens *both* among the pseudo documents as well as among the candidate synonyms. For example, in Figure 1, the tokens ‘ms’ and ‘spreadsheet’ are common among the candidates. We develop novel algorithms to exploit this overlap and avoid repeated work. Furthermore, we propose a MapReduce-based architecture for scalable discovery of synonyms and show how our algorithms can be incorporated into it (Section 5).

- **Handling long entity names with extraneous tokens:** Often times, entity catalogs contain long entity names which have many extraneous tokens. This makes it difficult in matching against query log, where queries are typically short. We propose search API based technique coupled with click information for finding more succinct strings of the entity to further improve the robustness of the framework (Section 6).

- **Experimental results:** We conduct extensive experiments using two real-life entity sets with different characteristics (location entities and software products). Our experiments show that (i) PseudoDocSim significantly outperform ClickSim and DocSim in terms of both precision and recall; (ii) our framework further improves the quality by enforcing the properties; (iii) our algorithms that share computation significantly outperform the baseline (by upto 6x); (iv) we demonstrate the usefulness of synonyms in improving search experience in real product settings (Section 7).

In this paper, we focus on entities with unambiguous reference strings and unambiguous candidate synonyms strings. Extending the framework to handle ambiguous reference strings and ambiguous candidate strings is an item of future work.

2. DEFINITIONS AND FRAMEWORK

In this section, we first define the synonym discovery problem and suggest a set of properties to define the synonym relation conceptually. Then, we present the notion of synonym similarity function and describe the general framework for discovering synonyms.

2.1 Synonym Relation and Properties

For a given entity e , let r_e denote the entity reference string for the entity e . Let S_e be the set of candidate synonym strings of entity e . Let $s_e \in S_e$ be a candidate synonym string of entity e . We first try to define the synonym relation on entity reference string r_e and candidate synonym strings S_e . We use the notation $r_e \equiv_{syn} s_e$ to denote that s_e is a synonym of r_e . In this work, we focus on unambiguous entity reference strings and synonym strings. An entity reference string r_e or a synonym string s_e is considered unambiguous if it uniquely identifies entity e . The synonym discovery problem is defined as follows in terms of input and output:

DEFINITION 1. (Synonym Discovery Problem) Given a reference entity string r_e of entity e and a set of candidate synonym strings S_e , identify all candidate synonym strings $s_e \in S_e$ such that $r_e \equiv_{syn} s_e$.

What desired properties should r_e and s_e have in order for $r_e \equiv_{syn} s_e$ to hold true? As motivated in Section 1, we believe the following properties should be true for judging $r_e \equiv_{syn} s_e$:

PROPERTY 1. (Reflexivity) $r_e \equiv_{syn} r_e, s_e \equiv_{syn} s_e$

This property states that a string (either entity reference string, or candidate synonym string) is always a synonym of itself. *The meaning of this property is that the strings in the synonym relation are unambiguous, which is the scope of this study.* If a string could refer to multiple different entities, then reflexivity does not hold.

PROPERTY 2. (Symmetry) if $r_e \equiv_{syn} s_e$, then $s_e \equiv_{syn} r_e$.

This property states that if s_e is synonym to r_e as the entity reference string, then r_e is also synonym to s_e as the entity reference string. One key property of synonym relation is that it has to be *two-way*, which can be inferred from the symmetry property. One way checking of synonym from s_e to r_e often leads to poor precision; this was also observed in [2]. We will specifically show how this property leads to the design of our synonym discovery framework, which ensures high precision.

PROPERTY 3. (Similarity) $r_e \equiv_{syn} s_e$, iff r_e and s_e are strongly related and both belong to the same concept class.

This property requires two important aspects. First, entity reference string r_e and candidate synonym string s_e needs to be highly related with each other in order to be synonymous. Further, r_e and s_e should both belong to the same concept class. This is used to overcome the current approaches' limitation: inability to distinguish entities of different classes. It is important to notice that these two aspects need to be judiciously *combined* when synonym checking is performed.

These natural synonym properties allow us to more systematically study the limitation of current techniques, in that they do not fully satisfy these properties.

2.2 Synonym Similarity Function

The challenge is to instantiate the (r_e, s_e) pairs that satisfy the similarity property. We need evidence to check the similarity property; specifically, we need evidence of whether they are strongly related and whether they belong to the same concept class. We resort to synonym similarity functions for this purpose.

²We use previously proposed techniques to generate the set of candidate synonym strings S_e of any entity e . Specifically, any query that shares at least one common clicked document with the reference entity string is a candidate synonym string [6].

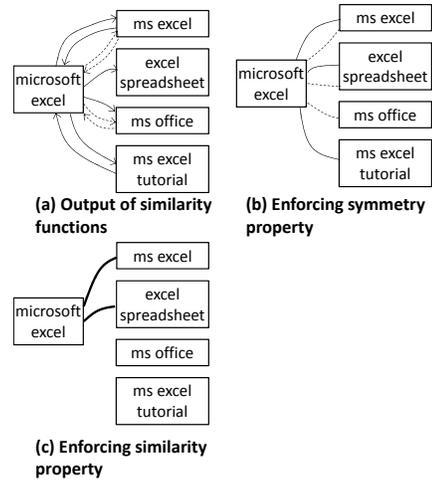


Figure 2: Synonym Discovery Steps

These functions serve as the starting point in populating the entire synonym relation between r_e and S_e . Figure 2 illustrates the 3 steps needed to discover synonyms, with Figure 2(a) focusing on instantiating relationship values based on two similarity functions (say \mathcal{F}_1 and \mathcal{F}_2). In the figure, solid thin edges represent relationship strength output from \mathcal{F}_1 and dashed edges represent relationship strength output from \mathcal{F}_2 ; thick edges refer to synonym relationship \equiv_{syn} .

As observed in prior works, the strings s_e and r_e alone are not enough for determining synonym relationship because string similarity between s_e and r_e is often not adequate [4, 3]. *As a result, we need to resort to auxiliary evidence for s_e and r_e .* Let $aux(s)$ denote the auxiliary evidence associated with a string s . One example auxiliary evidence $aux(s)$ associated with a string s can be the set of documents clicked by users for the web search query s or the set of documents in which s is mentioned.

Let a and b denote two strings.

DEFINITION 2. (Synonym Similarity Function) A synonym similarity function $\mathcal{F}(a \rightarrow b)$ (\mathcal{F} in short) takes input string a and its auxiliary evidence $aux(a)$ as well as input string b and its auxiliary evidence $aux(b)$, and computes the strength of relationship of a to b , formally:

$$\mathcal{F} : \{a, aux(a)\} \times \{b, aux(b)\} \rightarrow \mathcal{R} \in [0, 1]$$

Note that the function needs access to auxiliary evidence (e.g. web documents or click logs). We omit those inputs from the notation $\mathcal{F}(a \rightarrow b)$ for simplicity. Intuitively, b has strong relationship to a if the auxiliary evidence $aux(b)$ of b that supports b is related to a is significant compared with the auxiliary evidence $aux(a)$ that does not support it. Note that this function does not need to be symmetric.

2.3 Synonym Discovery Framework

Given the available synonym similarity functions, a general framework is needed for discovering entity synonyms to make sure that the synonym relation properties are satisfied. Figure 3 shows the synonym relation discovery framework. We need to judiciously combine multiple synonym functions to satisfy the similarity property. In general, the framework can take a wide variety of *synonym similarity functions* $\mathcal{F}_1, \dots, \mathcal{F}_n$ that can be used for checking synonym relationship. As a result, many existing useful similarity functions can be incorporated in this framework. Note that the similarity functions do not need to satisfy symmetry.

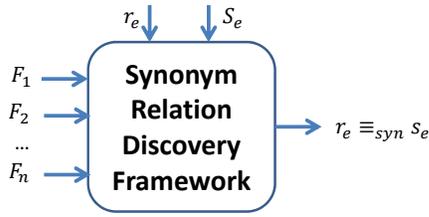


Figure 3: Synonym Discovery Framework

The framework can either be threshold based, or classifier based, which all perform checking based on the values output by synonym functions. This paper mainly focuses on a threshold based framework.

Synonym functions are not necessarily symmetric in that $\mathcal{F}(a \rightarrow b)$ may have different value from $\mathcal{F}(b \rightarrow a)$. We now discuss how our framework ensures symmetry. To ensure the symmetry property of synonym, the framework must perform *two-way checking* for asymmetric functions. In a threshold based framework (with threshold θ), we must make sure that the following condition is met to ensure symmetry:

$$\mathcal{F}(s_e \rightarrow r_e) \geq \theta \wedge \mathcal{F}(r_e \rightarrow s_e) \geq \theta \quad (1)$$

The above two-way checking states that s_e is a synonym of e if (i) s_e has strong relationship to r_e and (ii) r_e has strong relationship to s_e . As shown in Figure 2(b), by enforcing symmetry we can filter out candidates which only satisfy similarity one way (e.g., “ms office”).

This two-way checking in Eq. 1 is essential for synonyms, i.e., the relationship has to be strong from both directions. As observed in [2, 6], this enables us to distinguish true synonyms from other semantically related terms (e.g., other “nyms” like hypernym and hyponyms). In general, the measure of strength is *directional* and both direction needs to be checked explicitly (e.g., in PseudoDocSim proposed in Section 4). In some cases, it might be possible to obtain a single similarity function that captures both $\mathcal{F}(s_e \rightarrow r_e)$ and $\mathcal{F}(r_e \rightarrow s_e)$. For example, when the similarity is based on set intersection, the Jaccard Similarity can capture both (e.g., QCSim). Another example is PMI when the similarity is based on correlation. In such cases, one check is sufficient.

The similarity property states that the two aspects of synonym, relatedness and same class relationship, need to be checked in *combination*. This requires that, among the *synonym similarity functions* $\mathcal{F}_1, \dots, \mathcal{F}_n$ in the framework, there must exist at least one function for checking relatedness, and at least one function for making sure of the same class relationship.

As shown in Figure 2(c), by enforcing similarity property by judicious combination of various similarity functions, we can filter out candidates with only one aspect of the similarity property (e.g., “ms excel tutorial”).

Being able to accommodate various similarity functions is an advantage of our framework, as compared to single similarity function based approaches. This enables us to support the two key aspects of the similarity property, and therefore achieve higher precision.

3. BASELINE SIMILARITY FUNCTIONS

In this section, we study two previously proposed approaches, based on two existing similarity functions, ClickSim and DocSim, for finding synonyms and discuss why they are inadequate.

3.1 Click Similarity

In ClickSim, proposed in [6], the documents clicked for the web search query s is the auxiliary evidence $aux(s)$ associated with s . We first consider the strength $\mathcal{F}_{csim}(s_e \rightarrow r_e)$ of the relationship of s_e to r_e . The supporting evidence is the number of documents $|aux(s_e) \cap aux(r_e)|$ clicked for web search query r_e that are also clicked for s_e . The non-supporting evidence is the number of documents $|aux(r_e)| - |aux(s_e) \cap aux(r_e)|$ clicked for r_e but not clicked for s_e . The *strength* of the relationship of s_e to r_e is the significance of the former compared with the latter and formalized as follows:

$$\mathcal{F}_{csim}(s_e \rightarrow r_e) = \frac{|aux(s_e) \cap aux(r_e)|}{|aux(r_e)|}$$

Reversely, checking the relationship the other way around indicates the *exclusiveness* from s_e to r_e

$$\mathcal{F}_{csim}(r_e \rightarrow s_e) = \frac{|aux(r_e) \cap aux(s_e)|}{|aux(s_e)|}$$

ClickSim adjudges s_e to be a synonym of e iff

$$\mathcal{F}_{csim}(s_e \rightarrow r_e) \geq \theta \wedge \mathcal{F}_{cs}(r_e \rightarrow s_e) \geq \theta$$

As we can see, the formalization in Eq.1 accommodates ClickSim by capturing both strength and exclusiveness. Note that this approach already ensures the symmetry property. However, it does not meet the similarity property, in that it does make sure that candidate string is of the same class of the reference entity string.

3.2 Document Similarity

In the original document similarity proposed in [18], the documents which s occurs in is the auxiliary evidence $aux(s)$ associated with s . Turney just considered one-way checks; he only considered the strength $\mathcal{F}_{dsim}(r_e \rightarrow s_e)$ of the relationship of r_e to s_e . The supporting evidence is the number of documents $|aux(r_e) \cap aux(s_e)|$ that both s_e and r_e occurs in; the non-supporting evidence is the number of documents $|aux(s_e)| - |aux(r_e) \cap aux(s_e)|$ that s_e occurs in but not r_e . As in ClickSim, the strength of the relationship of r_e to s_e is formalized as follows:

$$\mathcal{F}_{dsim}(r_e \rightarrow s_e) = \log_2 \frac{|aux(r_e) \cap aux(s_e)|}{|aux(s_e)|}$$

s_e is adjudged a synonym of iff $\mathcal{F}_{ds}(r_e \rightarrow s_e) > \theta$. Note that this function is not symmetric. It does not meet the similarity property either, by failing to check same class relationship.

4. NOVEL SIMILARITY FUNCTIONS

As discussed in Section 3, both ClickSim and DocSim do not satisfy the synonym properties. Further, ClickSim often misses valid synonyms due to the sparsity of the click logs and DocSim suffers from noise in document content. In this section, we propose two novel similarity functions, namely *Pseudo Document Similarity* and *Query Context Similarity* to ensure the synonym properties and overcome the limitations.

4.1 Pseudo Document Similarity

We propose pseudo document similarity (PseudoDocSim in short) to address the sparsity problem. The main insight is that a document d can be concisely and accurately captured by the set of queries which clicked on it. *As a result, although a tail query q may not click on a document d , we can infer whether q is related to d through the other queries that clicked on d .* Recall the example in Figure 1, “ms spreadsheet” did not click on d_1 but “microsoft spreadsheet” and “ms excel” did. Since its tokens are “covered” by the tokens of the latter two queries, we can infer d_1 is related to “ms spreadsheet”. PseudoDocSim leverages this additional inferred evidence to overcome the sparsity problem of ClickSim.

To check whether a candidate synonym string is covered by queries clicked on a document, we construct the *pseudo document* for each document. The *pseudo document* of a document d (referred to as pseudodoc in short) is the set of all tokens from all the queries that clicked on document d .

DEFINITION 3. (**Pseudo Document**) Given a query click log L , the pseudo document of document d $PseudoDoc(d)$ is defined as: $PseudoDoc(d) = \{w|w \in q, s.t. q \text{ clicked on } d \text{ in log } L\}$

We typically use a query click log collected over a long period of time to construct comprehensive pseudo documents. For robustness, a minimal support (e.g., 5 clicks) is needed for a query, document pair to be included in the query click log.

The auxiliary information $aux(s)$ of string s is the set of documents clicked by query s . Formally, $aux(s) = \{d|s \text{ clicked on } d\}$

Consider a document $d \in aux(r_e)$. If d 's pseudodoc contains all the tokens in s_e , it is evidence supporting r_e is related to s_e ; otherwise, it is non-supporting evidence. The strength of the relationship of s_e to r_e , $F_{pdsim}(s_e \rightarrow r_e)$, is the fraction of documents in $aux(r_e)$ whose pseudo document contains all the tokens in s_e .

$$F_{pdsim}(s_e \rightarrow r_e) = \frac{|\{d|s_e \in PseudoDoc(d), d \in aux(r_e)\}|}{|aux(r_e)|} \quad (2)$$

Given Eq. 1, since PseudoDocSim is not symmetric in nature, we can enforce the two way checking by checking $F_{pdsim}(r_e \rightarrow s_e)$ in addition to $F_{pdsim}(s_e \rightarrow r_e)$, as follows:

$$F_{pdsim}(r_e \rightarrow s_e) = \frac{|\{d|r_e \in PseudoDoc(d), d \in aux(s_e)\}|}{|aux(s_e)|} \quad (3)$$

We refer to $F_{pdsim}(s_e \rightarrow r_e)$ and $F_{pdsim}(r_e \rightarrow s_e)$ as C-to-E (candidate-to-entity) and E-to-C (entity-to-candidate) similarities respectively. Similar enforcement can be applied to DocSim to ensure symmetry as well.

We now show that this two way checking of PseudoDocSim leads to higher recall than ClickSim as proved below.

LEMMA 1. (**Improved Recall**) For the same threshold, pseudo document similarity has higher recall compared with click similarity.

Proof: Consider a document $d_{common} \in (aux(s_e) \cap aux(r_e))$. Since s_e clicked on d_{common} , s_e is contained in d_{common} 's pseudo document, i.e., $s_e \subseteq PseudoDoc(d_{common})$. Hence, $d_{common} \in |\{d|s_e \in PseudoDoc(d), d \in aux(r_e)\}|$
 $\Rightarrow (aux(s_e) \cap aux(r_e)) \subseteq |\{d|s_e \in PseudoDoc(d), d \in aux(r_e)\}|$
 $\Rightarrow \frac{|(aux(s_e) \cap aux(r_e))|}{|aux(r_e)|} \leq \frac{|\{d|s_e \in PseudoDoc(d), d \in aux(r_e)\}|}{|\{d|d \in aux(r_e)\}|}$
 $\Rightarrow F_{csim}(s_e \rightarrow r_e) \leq F_{pdsim}(s_e \rightarrow r_e)$.

This implies $F_{csim}(s_e \rightarrow r_e) \geq \theta \Rightarrow F_{pdsim}(s_e \rightarrow r_e) \geq \theta$. Exactly in the same way, we can show that $F_{csim}(r_e \rightarrow s_e) \geq \theta \Rightarrow F_{pdsim}(r_e \rightarrow s_e) \geq \theta$. Hence, $F_{csim}(s_e \rightarrow r_e) \geq \theta \wedge F_{csim}(r_e \rightarrow s_e) \geq \theta \Rightarrow F_{pdsim}(s_e \rightarrow r_e) \geq \theta \wedge F_{pdsim}(r_e \rightarrow s_e) \geq \theta$ ■

There are two main benefits of using pseudo document similarity. *First*, it harvests strictly more supporting evidence than ClickSim. if $d \in aux(r_e)$ is supporting evidence of s_e in ClickSim (i.e., r_e clicked on d), it is also supporting evidence in pseudodoc. Even if d is not supporting evidence in ClickSim, it can serve as supporting evidence in PseudoDocSim based on additional inferred information.

Second, in contrast to DocSim, pseudo document allows us to focus on the essential parts of a document, rather than the complete content. Consider the example in Figure 1. Although document d_1 has many other words in its content, its pseudo document only contains words “microsoft”, “spreadsheet”, “ms” and “excel”: a very succinct yet high quality representation of the document. Mining over pseudo documents yields higher precision, as compared to the document similarity approach. Furthermore, since pseudo documents are much shorter than real documents, it is much more efficient to compute as well.

4.2 Query Context Similarity

We propose query context similarity (QCSim in short) to make sure that candidate synonym string is of the same class of the entity. Since query strings are considered as candidate synonym strings, ClickSim often includes candidate synonym strings of other concept classes as synonyms. For example, “ms excel tutorial” could be adjudged as synonym for “microsoft excel” because they are strongly related to each other based on our auxiliary evidence. The technical challenge is to filter out candidate synonym strings that are of different concept classes.

Our main insight here is that the words that appear in the context of entity names in web search queries can help us distinguish between entities of different classes. These words describes the various facets of the concept class, and as a result tend to be similar for entities from the same class and different for entities from different classes. Contexts of a string are the additional words (consisting of 1, 2, 3 words) occur immediately on its left and right in web search queries; we require the number of occurrences to be above a threshold to eliminate noise. For example, “microsoft excel” and its true synonyms like “microsoft spreadsheet”, “ms excel” and “ms spreadsheet” have contexts like “download”, “help”, and “training” while “ms excel tutorial” have (very different) contexts like “book”, “ppt” and “guide”. We compute QCSim between r_e and s_e by taking the set similarity of their contexts. We use Jaccard Similarity in this paper; any other measure of set similarity (e.g., Cosine, Dice coefficient, etc.) can be used.

The auxiliary information $aux(s)$ of a string is the set of contexts in web search queries. In this case, we use a single similarity function to capture both $\mathcal{F}(s_e \rightarrow r_e)$ and $\mathcal{F}(r_e \rightarrow s_e)$:

$$F_{qcsim}(s_e \rightarrow r_e) = F_{qcsim}(r_e \rightarrow s_e) = \frac{|aux(s_e) \cap aux(r_e)|}{|aux(s_e) \cup aux(r_e)|} \quad (4)$$

Here, the two way checking is automatically satisfied.

Note that QCSim is very similar to distributional similarity using contexts [11, 14, 17]. However, most prior work uses contexts in documents; to the best of knowledge, this is the first time distributional similarity has been applied to contexts in web search queries.

4.3 Combining the Two Similarity Measures

Formally, the synonym discovery process in our framework can be stated as follows:

DEFINITION 4. (**Synonym Discovery Process**) Given a reference entity string r_e of entity e and a set of candidate synonym strings S_e , identify all $s_e \in S_e$ such that

- (i) $F_{pdsim}(s_e \rightarrow r_e) \geq \theta_1 \wedge F_{pdsim}(r_e \rightarrow s_e) \geq \theta_1$ and
- (ii) $F_{qcsim}(s_e \rightarrow r_e) (= F_{qcsim}(r_e \rightarrow s_e)) \geq \theta_2$

We follow the above approach in this paper. In practice, such hard thresholds on each similarity measure are difficult to determine and may be too restricting; a more practical approach is to use the similarity values as features and use a classifier to determine whether r_e and s_e are synonyms.

5. EFFICIENT AND SCALABLE ALGORITHMS

Often, we need to generate synonyms for millions of entities, and therefore efficiency and scalability become critical. We first present the basic system architecture. Further, we observe that there is a high degree of overlap in computation; we develop algorithms that share computation and avoid repeated work. Finally we show how we can leverage the MapReduce framework to generate synonyms at such large scale.

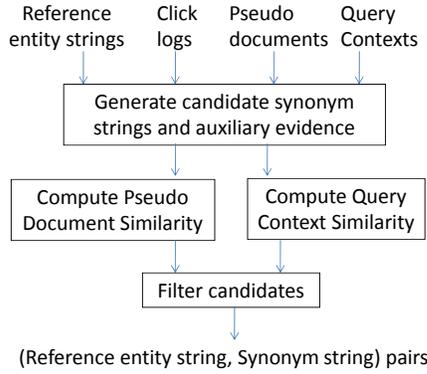


Figure 4: System Architecture

5.1 System Architecture

The architecture of our system is shown in Figure 4. We assume the pseudo documents for all documents (that have non-zero clicks in the click log) are computed in advance; this is a one-time job that can be reused for any reference entity set.³ We also assume that the contexts of all query strings in the query log (in longer queries) are also computed in advance. Our architecture has 4 steps:

- 1) **Generation of Candidates and Auxiliary Evidence:** Given the reference entity strings and the click logs, this step generates (i) the candidate synonym strings of each entity and (ii) the auxiliary evidence for each reference entity string and each candidate synonym string.
- 2) **Pseudo Document Similarity Computation:** Given the (entity, candidate) pairs and the pseudodocs for both the reference entity strings and the candidates, this step computes the E-to-C similarity as well as C-to-E similarity based on Equations 2 and 3.
- 3) **Context Similarity Computation:** Given the (entity, candidate) pairs and the query contexts, this step computes the context similarity for each pair based on Equation 4.
- 4) **Filtering:** Given the (entity, candidate) pairs and their E-to-C similarities, C-to-E similarities and context similarities, this step filters out the non-synonyms and outputs the final (entity, synonym) pairs.

The most expensive computation component in the architecture is the PseudoDocSim computation, and as a result becomes the bottleneck of the framework. We focus on this step in the rest of the section.

5.2 PseudoDocSim Computation

We consider computing the C-to-E similarities; E-to-C similarities are computed in exactly the same way. There are two main decisions: how to save the computation cost of calculating PseudoDocSim (efficiency) and how to partition the task into subtasks (scalability).

5.2.1 Efficient Computation

We present three algorithms to calculate PseudoDocSim: a baseline algorithm for containment checking that does not exploit any overlap of tokens, an algorithm, called DocIndex, that exploits overlap on the document side but not on the candidate side and finally,

³The pseudo documents need to be recomputed when the information in the click logs have changed significantly. Since we are using click logs collected over multiple years (say, 2 or 3 years), we anticipate this to happen once every several months (say, once in six months). Hence, we do not consider the cost of constructing pseudo documents as part of the synonym generation task.

an algorithm, called DualIndex, that exploits overlap on both document and candidate sides.

The task performed by the algorithms can be formally defined as follows.

DEFINITION 5. (PseudoDocSim Computation Task) Given the reference entity string r_e , its pseudo documents $D(r_e)$ and its candidate synonym strings S_e , compute E-to-C similarities between the reference entity string r_e and each candidate $s_e \in S_e$.

Baseline Algorithm: The naive algorithm is, for each candidate s_e and each pseudo document d_e , to check the containment of tokens in s_e in d_e . We tokenize the each pseudo document only once and insert the tokens of each pseudo document in a separate hash table to allow containment checking.

Cost analysis: Let $|T(s)|$ denote the number of tokens in the string s . The number of hash table lookups for candidate s_e is $|T(s_e)| \times |D(r_e)|$. The overall number of hash table lookups is $\sum_{s_e \in S_e} |T(s_e)| \times |D(r_e)|$. The total cost is $C_h \times \sum_{s_e \in S_e} |T(s_e)| \times |D(r_e)|$ where C_h is the average cost of a hash lookup.

The naive algorithm is expensive as it performs too many hash lookups. For example, if an entity has 1000 candidates and 1000 representative documents and a candidate has an average of 5 tokens, the number of lookups is $5 * 1000 * 1000 = 5$ million. We observe that there is a high degree of overlap of tokens both among the pseudo documents in $D(r_e)$ and the candidates in S_e ; it is possible to reduce the cost by exploiting this overlap.

Inverted Index on Pseudo Documents: We propose to build an inverted index on the pseudo documents. For each candidate s_e , we obtain the pseudo documents containing the tokens in s_e using the inverted index, i.e., by intersecting the posting lists corresponding to the tokens in s_e . We assume that the inverted index fits in memory; this is realistic since it is built on pseudo documents (which are short) of a single entity.

Cost analysis: We assume that the cost of accessing the inverted index and intersecting the posting lists for each candidate s_e is proportional to the number of tokens $|T(s_e)|$ in s_e , i.e., it is $C_l \times |T(s_e)|$ where C_l is the average access/intersection cost per token. The overall cost is $C_l \times \sum_{s_e \in S_e} |T(s_e)|$.

The above algorithm, referred to as *DocIndex*, is an improvement over the naive algorithm as it exploits the overlap of tokens among the pseudo documents. A token is looked up just one for each candidate compared with $|D(r_e)|$ times in the naive case. We observe that it is possible to further improve cost by exploiting overlap of tokens among candidates as well.

Inverted Index on both Pseudo Documents and Candidates: In addition to the inverted index on the pseudo documents, we propose to build an inverted index on the candidates. Subsequently, the algorithm iterates over each distinct token in the set S_e of candidates and constructs the Candidate-Document matrix (CD matrix) as follows. We first assign an index $ind(s_e)$ between 1 and $|S_e|$ to each candidate and an index $ind(d_e)$ between 1 and $|D_e|$ to each representative document. The (i, j) th cell $CD[i, j]$ in the CD matrix contains the number of tokens of the candidate with index i contained in the pseudo document with index j . We assume that the two inverted indexes and the CD matrix fits in memory.

Initially, all cells in the CD matrix are set to 0. For each distinct token t in S_e , we lookup the inverted indexes on pseudo documents and candidates. For each document d and each candidate s containing t , we increment $CD[ind(d), ind(s)]$ by 1. The CD matrix computation is completed when we have iterated over all distinct tokens in S_e .

After the CD matrix has been computed, we compute the E-to-C similarity between r_e and a candidate s_e as follows. We count the

Algorithm 1 DUALINDEX

Input: $S(e), D(e)$ **Output:** E-to-C similarities $\forall i, j$ CDMatrix[i,j] $\leftarrow 0$

```
1: if  $|S(e)| \leq |D(e)$  then
2:   for  $t \in DisTok(S(e))$  do
3:      $S_t \leftarrow$  Index of candidates containing  $t$ 
4:      $D_t \leftarrow$  Index on pseudo documents containing  $t$ 
5:     for  $i \in S_t$  do
6:       for  $j \in D_t$  do
7:         CDMatrix[i,j]  $\leftarrow$  CDMatrix[i,j] + 1
8:   else
9:     for  $t \in DisTok(D(e))$  do
10:      // Same as lines 2-5
11: for  $s \in S(e)$  do
12:    $E - to - C - Sim(e, s) \leftarrow \frac{|d|d \in D(e) \wedge CD[ind(s), ind(d)] = |s||}{|D(e)|}$ 
13: return  $E - to - C - Sim(e, s)$ 
```

number of documents $d_e \in D_e$ which contains all the tokens of s_e , i.e. for which $C[ind(s_e), ind(d_e)] = |T(s_e)|$. Dividing that count by the number of pseudo documents yields the E-to-C similarity.

Cost analysis: We assume the cost of accessing the two inverted indexes is proportional to the number of tokens it is accessed for. We assume the cost of updating the matrix is very efficient because it is direct access. Let $DisTok(S_e)$ denote the set of distinct tokens in the set S_e of candidates and $DisTok(D_e)$ denote the set of distinct tokens in the set D_e of candidates. The overall cost is therefore $C_a \times \min(|DisTok(S_e)|, |DisTok(D_e)|)$ where C_a is the average access cost per token.

The above algorithm, referred to as *DualIndex*, further improves the cost over DocIndex by exploiting the overlap among candidates as well. A token t is looked up just once in DualIndex as opposed to $|C(t)|$ times where $|C(t)|$ is the number of candidates that contains that token.

5.2.2 Partitioning the Task

To scale the computation of PseudoDocSim, we leverage the MapReduce framework to partition the computation across various distributed nodes, which is achieved by specifying the appropriate key to the Map step. We propose to partition the computation into subtasks by entity. Each subtask computes the E-to-C similarities between a reference entity string and all its candidate synonym strings. Formally, the Map and Reduce steps are (the key is r_e):

Map: $\langle r_e, s_e, d_e \rangle \rightarrow \langle r_e, \langle s_e \rangle, \langle d_e \rangle \rangle$ Reduce: $\langle r_e, \langle s_e \rangle, \langle d_e \rangle \rangle \rightarrow \langle r_e, \langle s_e, \mathcal{F}_{pdsim}(r_e \rightarrow s_e) \rangle \rangle$

The Map step groups the set of candidate synonym string into $\langle s_e \rangle$, and the set of pseudo documents for r_e into $\langle d_e \rangle$. The reduce step performs the PseudoDocSim computation of all candidates with respect to r_e and output the $\mathcal{F}_{pdsim}(r_e \rightarrow s_e)$ value for each candidate string s_e .

Adapting the other steps in the system architecture to the MapReduce framework is straightforward. We skip the details to save space.

6. HANDLING LONG ENTITY NAMES

Often times, entity catalogs contain long entity names which have many extraneous tokens. Matching such long entity names directly against query log often yields very few or no queries, which makes subsequence synonym mining difficult or even impossible. To make the framework robust, can we identify the key tokens in a long entity name, and discard extraneous tokens, so that we can better match the entity with queries in query log?

It is common to have very long entity names which contain peripheral information about the entity. For instance, camera names in a camera catalog often includes spec information (e.g., megapixel, zoom, etc.). One such example is “Canon EOS 350D - digital camera, 8MP, 3x Optical Zoom”. To deal with such cases, we propose to leverage search engine API to find proper subset of tokens (subset synonyms) that can still uniquely identify the entity.

Specifically, given such an entity reference string r_e , we first use search engine API to retrieve a set of documents by using the entity reference string as the query, and treat these documents as the auxiliary information $aux(r_e)$ for r_e . A query which clicked on at least one of the documents in this set, whose tokens is a subset of those in r_e , is regarded as a candidate s_e . Then we can perform click similarity analysis based on strength of relationship and exclusiveness, as described in Section 3.1, to find the subset synonyms. For instance, we may find that “Canon EOS 350D” is a qualifying subset synonym of “Canon EOS 350D - digital camera, 8MP, 3x Optical Zoom” using this analysis. It is worth noticing that here we are restricting to subsets only, which makes click similarity analysis more accurate. Further, pseudo document similarity does not work here since extraneous tokens such as “8MP, 3x Optical Zoom” often do not appear in pseudo documents.

Once we found such subset synonym of the entity, we can use it as the reference string of the entity and perform synonym discovery.

7. EXPERIMENTAL EVALUATION

In this section, we report our experiment results. We first describe the setting of our experiments. Then we report the quality results to validate the effectiveness of our framework. We further report efficiency and scalability results. Finally, we discuss the impact of the generated synonyms in improving search.

7.1 Experimental Setting

Our synonym discovery framework mainly leverages query log. Specifically, we leverage the query click log from “Bing” from 2009 to 2010. Note that due to proprietary and privacy concerns we cannot share all the details of the query click log.

To empirically evaluate the framework, we conduct experiments on two real life datasets, as summarized below:

- **Local Business Names (Local):** 937 local business names sampled from “Bing” local catalog, e.g., *la police credit union*;
- **Software Names (Software):** 10 software names, e.g., *microsoft excel*.

These two datasets, from two very different domains, are of very different characteristics. The local business names dataset, by its nature, tend to have niche entities. As a result, they do not have extensive appearance in query click log. Meanwhile, the entities from the software names datasets tend to be more popular. There are often a large number of click data available for such entities. The discovered synonyms are judged by human experts as to whether they are true synonyms or not.

We implemented our synonym discovery framework on top of COSMOS, a distributed MapReduce framework based on Dryad [12].

7.2 Quality Results

To give an intuitive feeling of the output, a few example entities with entity synonyms discovered by our framework are listed in Table 1.

Next, we systematically study the performance of our framework, and compare it to the state-of-the-art. Specifically, we report the results on the following 4 settings:

- (I) ClickSim: click similarity (Section 3)

Domain	Entity	Synonyms
Local	la police credit union	lapfcu, los angeles police fcu, ...
Local	bemidji state university	bemidji state u, bemidji state univ, ...
Software	microsoft excel	ms excel, spreadsheet excel, microsoft office excel, ...
Software	microsoft sql server	ms sql server, sqlserver, mssql, ...

Table 1: Example Entity Synonyms Discovered

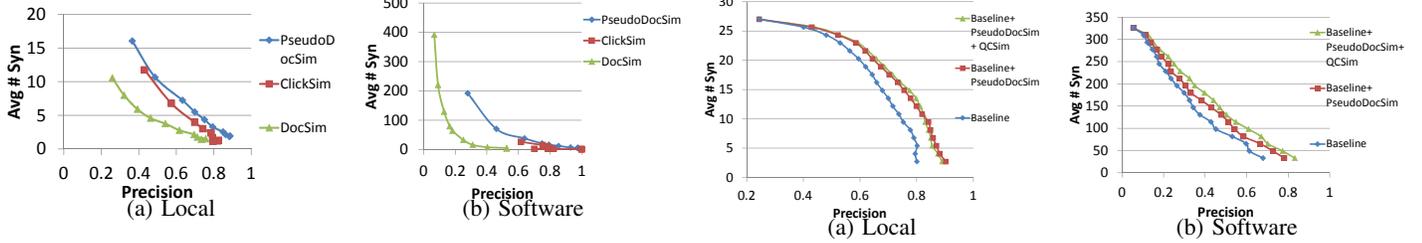


Figure 5: Quality Results (Threshold)

- (II) DocSim: document similarity (Section 3)
- (III) PseudoDocSim: pseudo document similarity (Section 4)
- (IV) PseudoDocSim+QCSim: pseudo document similarity with context similarity (Section 4)

We use two main measures for effectiveness evaluation:

- Precision: number of true synonyms divided by the total number of synonyms output;
- AvgNumOfSynonym: Average Number of Synonyms Per Entity.⁴

We first examine the performance of PseudoDocSim when compared against ClickSim and DocSim. We vary the threshold value from 0 to 1.0 to examine the precision and recall tradeoff. Figure 5(a) and (b) report the results on Local and Software datasets respectively. As we can see, PseudoDocSim consistently outperforms ClickSim by outputting more number of synonyms per entity, while still maintaining high accuracy. This validates our design in that PseudoDocSim overcomes the sparsity problem of ClickSim to output more synonyms. Interestingly, it also significantly outperforms DocSim in both precision and recall. The improvement in precision is expected, since DocSim suffers from the noise from document content. The improvement in recall is due to the fact that search engines’ ability to direct a query to a document (by query alternation, spell checking or partial match) although the query does not appear exactly in the document. Since PseudoDocSim leverages query log, it could harvest such rich signals. Comparing the results across the two domains, we also find that the software domain tend to have much higher number of synonyms generated per entity. This is due to the diverse characteristics of the two domains as discussed. The results show that we can generate ~ 12 synonyms per entity at $\sim 85\%$ precision for software entities, and ~ 4 synonyms per entity at $\sim 80\%$ precision for local entities.

We also study how the various similarity features contribute to the output result in using a classifier. We first examine the classification results over baseline (using only ClickSim and DocSim), then we study how the results change when we add in PseudoDocSim and QCSim as features. Figure 6(a) and (b) show the precision and recall tradeoff over the two domains, by using a standard boosted tree based classifier. As we can see, adding PseudoDocSim feature significantly improves the performance over baseline, and

⁴it is almost impossible to know the universal set of synonyms for each entity, and therefore hard to report the traditional recall numbers. As an alternative, we report the average number of synonyms generated per entity.

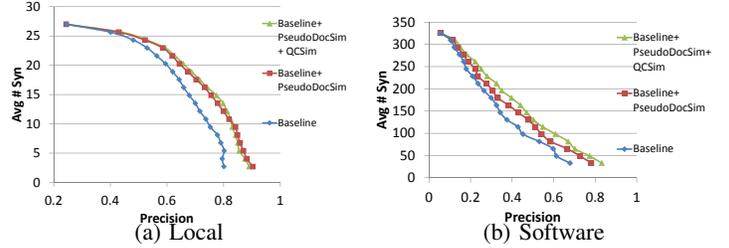


Figure 6: Classification Results

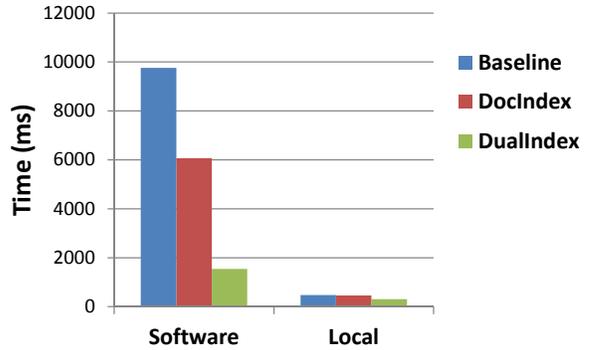


Figure 7: Efficiency Results

adding QCSim further improves the quality. The effect of QCSim is relatively minor for the Local dataset. This is due to the fact that many entities in this dataset are tail entities, and therefore having very limited query context information available.

7.3 Efficiency Results

Now we report the processing efficiency results on the two real datasets. We report the total processing time with respect to the three schemes discussed in Section 5, referred to as *Baseline*, *DocIndex* and *DualIndex* respectively.

Figure 7 shows the overall processing time to calculate the PseudoDocSim score (the main computation of the framework) for the two datasets. As we can see, *DocIndex* consistently outperforms *Naive*, and *DualIndex* significantly improves over *DocIndex*. This is because *DualIndex* is able to fully exploit the overlap between candidates and pseudo documents, whereas *DocIndex* only leverage the overlap between pseudo documents and *Baseline* does not leverage any overlap. *DualIndex* is shown to offer more than 6x speedup over *Baseline* in the Software dataset, since entities in this dataset tends to have many more pseudo documents as well as candidates, as compared to the entities in the Local dataset. As a result, there is more overlap opportunities to exploit in the Software dataset, where *DualIndex* shines.

7.4 Impact on Improving Search

To see the impact of synonyms on search, we perform judgement on search results in two settings: setting (a) search over the original catalog versus setting (b) search over synonym augmented catalog. All other aspects of the two setting are identical.

Specially, expert judges first perform relevance judgement under setting (a) over a query set. The same judges then perform relevance judgement under setting (b) using the same query set. The classical normalized discounted cumulative gain (nDCG) is used to judge the quality of retrieved results.

Study was performed in two Bing products, Bing Shopping and Bing Video. 2000 queries were sampled from their perspective query logs, and used for relevance evaluation. For each query, the top 4 returned results are evaluated.

To get a concrete feeling, the table below shows a few examples, where we see the original entity name, the discovered synonym and the specific query the synonym helped in improving nDCG. As we can see, without the synonym, it would be hard to directly match the query with the entity name.

Entity Name	Synonym	Query
pbs kids play!	pbs kids org games	pbs kids.org
mitsubishi wd-73c9 73" rear projection TV	mitsubishi wd73c9	wd73c9

Overall, we see that the synonyms bring +0.1 nDCG (on 0-100 scale) in Bing Shopping search. Notice, we obtain this gain in the actual production setting of Bing Shopping, which is a highly optimized engine with many existing features. More interestingly, we zoomed into the set of difficult queries (94 queries with 0 nDCG in setting (a)). We see an nDCG improvement of +0.8 over these queries, a significant boost. For Bing Video search, we witness +0.2 nDCG improvement overall with the help of synonyms.

As we can see, the discovered entity synonyms have clearly helped in both search scenarios, resulting in higher nDCG numbers and therefore better search experience.

8. RELATED WORK

Researchers have proposed several approaches to automatically discovering synonyms using web data [6, 7, 18, 2, 11, 17, 3, 5]. They include click similarity [6, 7], document similarity [18, 2] and distributional similarity [11, 17]. As we discussed in Section 1, none of the above approaches satisfies all the properties and suffers from other limitations like click log sparsity, inability to distinguish between entities of different concept classes and difficulty to scale. Techniques to generate a particular class of synonyms, ones that are substrings of the given entity names, have been proposed recently [3, 5]. Our focus is to generate *all* classes of synonyms.

There exists a rich body of work on identifying similar queries; this can be used for query suggestions, query alteration, document ranking and ad matching [13, 8, 15]. Many techniques have been proposed to compute query similarities. For example, Craswell and Szummer compute it by using random walks on the click graph [8], Jones et. al. do so based on typical substitutions web searchers make in their queries [13] while Mei et. al. use hitting time on the query click graph [15] (e.g., using random walk on the click graph). They are not applicable to synonym discovery as they compute all similar queries, not just synonyms.

Approximate string matching techniques (when applied between reference string and candidate string) can be used to detect some classes of synonyms [16]. This can discover synonyms due to different different normalization (e.g., “canon eos-400d” vs. “Canon EOS 400d”) or misspellings (e.g., “cannon eos 400d” vs. “Canon EOS 400d”), but not other types of variations like acronyms, semantic variations or subset/supersets.

Our work is related to the works on entity resolution (also known as entity conflation, reference reconciliation or record matching) which try to resolve difference references to the same real world entity [9]. They typically rely on a rich set of attributes to be present

to produce high quality results. Such information is not available in a web setting, hence those techniques cannot be directly applied.

Our work is also related to the traditional word synonym discovery problem from text in NLP [10, 19]. The synonyms we focused on this this work are entity synonyms. Unlike words, it is non-trivial to identify entities of any domain from text. Furthermore, many legitimate entity synonyms do not necessary appear as one phrase in text. Another difference is that this work mainly leverages query log for mining synonyms. The mining space is much more succinct than that of text, and therefore is more practical for mining entity synonyms at large scale.

9. CONCLUSION

In this paper, we proposed a general framework for discovering entity synonyms. We study novel similarity functions that overcome the limitations of previously proposed functions. We developed efficient and scalable algorithms to generate such synonyms, and robust techniques to handle long entity names. Our experiments demonstrate superior quality of our synonyms and efficiency of our algorithms, as well as its impact in improving search.

10. REFERENCES

- [1] S. Agrawal, K. Chakrabarti, S. Chaudhuri, and V. Ganti. Scalable ad-hoc entity extraction from text collections. *Proc. VLDB Endow.*, 2008.
- [2] M. Baroni and S. Bisi. Using cooccurrence statistics and the web to discover synonyms in technical language. In *In Proceedings of LREC 2004*, pages 1725–1728, 2004.
- [3] S. Chaudhuri, V. Ganti, and D. Xin. Exploiting web search to generate synonyms for entities. In *WWW Conference*, 2009.
- [4] S. Chaudhuri, V. Ganti, and D. Xin. Mining document collections to facilitate accurate approximate entity matching. *PVLDB*, 2009.
- [5] S. Chaudhuri, V. Ganti, and D. Xin. Mining document collections to facilitate accurate approximate entity matching. *PVLDB*, 2(1), 2009.
- [6] T. Cheng, H. Lauw, and S. Pappas. Fuzzy matching of web queries to structured data. In *ICDE*, 2010.
- [7] T. Cheng, H. W. Lauw, and S. Pappas. Entity synonyms for structured web search. *TKDE*, 2011.
- [8] N. Craswell and M. Szummer. Random walks on the click graph. In *SIGIR*, 2007.
- [9] X. Dong, A. Halevy, and J. Madhavan. Reference reconciliation in complex information spaces. In *SIGMOD*, 2005.
- [10] G. W. Furnas, S. C. Deerwester, S. T. Dumais, T. K. Landauer, R. A. Harshman, L. A. Streeter, and K. E. Lochbaum. Information retrieval using a singular value decomposition model of latent semantic structure. In *SIGIR*, 1988.
- [11] Z. Harris. Distributional structure. *Word*, 10(23), 1954.
- [12] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly. Dryad: distributed data-parallel programs from sequential building blocks. In *EuroSys*, 2007.
- [13] R. Jones, B. Rey, O. Madani, and W. Greiner. Generating query substitutions. In *WWW*, 2006.
- [14] C. D. Manning and H. Schütze. *Foundations of statistical natural language processing*. MIT Press, 1999.
- [15] Q. Mei, D. Zhou, and K. Church. Query suggestion using hitting time. In *CIKM*, 2008.
- [16] G. Navarro. A guided tour to approximate string matching. *ACM Comput. Surv.*, 2001.
- [17] P. Pantel, E. Crestan, A. Borkovsky, A.-M. Popescu, and V. Vyas. Web-scale distributional similarity and entity set expansion. In *EMNLP*, 2009.
- [18] P. D. Turney. Mining the web for synonyms: Pmi-ir versus lsa on toefl. *CoRR*, cs.LG/0212033, 2002.
- [19] T. Wang and G. Hirst. Near-synonym lexical choice in latent semantic space. In *COLING*, 2010.