LSH Ensemble: Internet-Scale Domain Search

Erkang Zhu
University of Toronto
ekzhu@cs.toronto.edu

Fatemeh Nargesian
University of Toronto
fnargesian@cs.toronto.edu

Ken Q. Pu
UOIT
ken.pu@uoit.ca

Renée J. Miller
University of Toronto
miller@cs.toronto.edu

ABSTRACT

We study *the problem of domain search* where a domain is a set of distinct values from an unspecified universe. We use Jaccard set containment score, defined as $|Q \cap X|/|Q|$, as the measure of relevance of a domain X to a query domain Q. Our choice of Jaccard set containment over Jaccard similarity as a measure of relevance makes our work particularly suitable for searching Open Data and data on the web, as Jaccard similarity is known to have poor performance over sets with large differences in their domain sizes. We demonstrate that the domains found in several real-life Open Data and web data repositories show a power-law distribution over their domain sizes.

We present a new index structure, Locality Sensitive Hashing (LSH) Ensemble, that solves the domain search problem using set containment at Internet scale. Our index structure and search algorithm cope with the data volume and skew by means of data sketches using Minwise Hashing and domain partitioning. Our index structure does not assume a prescribed set of data values. We construct a cost model that describes the accuracy of LSH Ensemble with any given partitioning. This allows us to formulate the data partitioning for LSH Ensemble as an optimization problem. We prove that there exists an *optimal* partitioning for any data distribution. Furthermore, for datasets following a power-law distribution, as observed in Open Data and Web data corpora, we show that the optimal partitioning can be approximated using equi-depth, making it particularly efficient to use in practice.

We evaluate our algorithm using real data (Canadian Open Data and WDC Web Tables) containing up over 262 million domains. The experiments demonstrate that our index consistently outperforms other leading alternatives in accuracy and performance. The improvements are most dramatic for data with large skew in the domain sizes. Even at 262 million domains, our index sustains query performance with under 3 seconds response time.

Proceedings of the VLDB Endowment, Vol. 9, No. 12

Country	Number of Datasets (Structured and Semi-Structured)
US	191,695
UK	26,153
Canada	244,885
Singapore	11,992

Table 1: Examples of Governmental Open Data as of First Quarter 2016.

1. INTRODUCTION

In the Open Data movement, large volumes of valuable databases are being published on the Web. Governments around the world are launching Open Data portals (some of which are shown in Table 1). The data format is highly heterogeneous, comprised of a mixture of relational (CSV and spreadsheet), semi-structured (JSON and XML), graph based (RDF), and geo-spatial formats. There is an increasing number of datasets in which well-structured attributes (with or without a name) can be identified, each containing a set of values that we will call a *domain*.

It is not just federal governments that are releasing massive numbers of datasets. Several projects have extracted tables from HTML pages [7, 18]. Cafarella et al. [7] extracted 154 million relational tables from Google's general-purpose web crawl. Lehmberg et al. [18] have compiled and published 51 million relational tables extracted from Common Crawl. Tables extracted by Lehmberg et al., called Web Data Commons (WDC) Web Tables, are open and accessible to the public outside of search engines. These tables provide a common ground for research on Data on the Web.

Despite the openness of these datasets, effectively gaining access to them is still challenging for many reasons.

- There is lack of schema description in most of the datasets. Identifying relevant datasets is exceedingly difficult.
- In most cases, the only form of data access is bulk download. This means traditional search at the data level is impossible at the moment.
- Data is hosted on the Web. While the Web is a robust and open platform, it provides very low bandwidth for data access. Database operations such as sorting, scan and join are nearly impossible over the Web. This makes *ad hoc* access of Open Data impractical.

1.1 Web Joins Using Domain Search

In this paper, we focus on a specific problem of Open Data management: the domain search problem. A *domain*

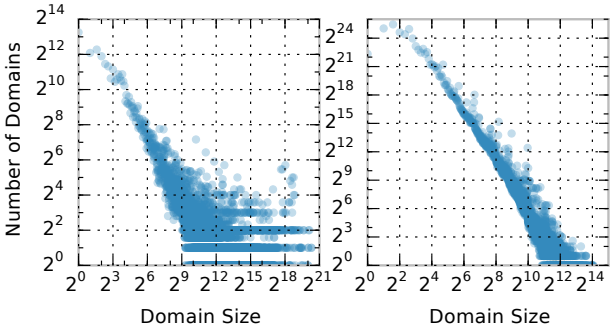

Figure 1: Domain Size Distribution of the Canadian Open Data (Left) and the English Relational Subset of WDC Web Table Corpus 2015 (Right).

is simply a set of values. A dataset can be characterized by one or more domains. For example, the Canadian federal research agency publishes information about the industry partners of successful research grants as relational tables. One such table `NSERC_GRANT_PARTNER_2011` has the following attributes:

```
Identifier, Partner, Province, Country, Fiscal Year, ...
```

Each attribute contains a domain. Given such a table, an interesting question is to find other tables that join with this table on a specified attribute. For example, to find additional information about industry partners a data scientist may wish to find tables that join on `Partner`. To do this, we must find tables that include a domain that **contains** as much of the `Partner` domain as possible.

We define the domain search problem as an R-near neighbor problem [2]. Given a query domain Q and a relevance threshold t^*, find all domains X whose relevance to Q is within t^*. The unique characteristics of web data impose constraints on a solution to the domain search problem.

- *Scalability.* For Internet-scale domain search, a solution must be able to handle hundreds of millions of domains.
- *Open world domains.* A solution cannot assume a fixed vocabulary or prescribed set of values covering all domains. As new domains are introduced (potentially with new, unseen values), the search should be able to handle them.
- *Skewed distribution.* Some domains are small (a domain of provinces has only a few values), while others can be quite large (the Canadian government's contract-disclosure dataset has domains with tens of thousands of distinct values). We have collected the Canadian Open Data repository and the English relational subset of WDC Web Tables Corpus 2015 [18]. Figure 1 shows the respective domain size distributions. It is clear that in each case, the domain sizes follow a power-law distribution.
- *Relevance.* For the domain search problem, a domain is relevant if it contains as much of the search domain as possible. Jaccard similarity (size of the intersection over the size of the union) is a commonly used measure of relevance between sets when the sets have comparable cardinalities. Open data and Web data domains have potentially large differences in their domain sizes. It is known [1] that Jaccard similarity is not an accurate or intuitive measure of relevance due to its bias to smaller domains. Related measures, including the cosine similarity or dice coefficient, in fact any measure that is a ratio of two functions of both domains ($\frac{f(x,y)}{g(x,y)}$) will have the same bias when applied over sets with a large skew in their cardinality. For the purpose of domain search, where our goal is to find domains that contain as much of the query domain as possible, it is more intuitive to define the relevant domains as the ones with *large* overlap with the query domain. Thus we choose the (Jaccard) *set containment* score as the relevance measure (the size of the intersection divided by the size of the query domain).
- *Compact Index Size and Small Query Memory Footprint.* In order to scale to Internet scale of hundreds of millions of domains, we need the index to be highly compact in size. We also want the representation of the search query domain to have small memory footprint as it needs to be exchanged over the Web. If we use raw values of the domains, the data exchange would degrade the query performance significantly due to the relatively low bandwidth of the Internet.

The scalability and compact index size limitations suggest using small, fixed size data sketches to effectively approximate a relevance score. The requirement for supporting open-world domains makes a hashing approach like the *Minwise Hashing* [6] data sketch, or simply MinHash, a natural choice to represent domains. It is known that the Jaccard similarity can be accurately estimated using MinHash. Furthermore, *Locality Sensitive Hashing* (LSH) is an efficient index for approximate R-near neighbor queries [2], and it can be used for the Jaccard similarity search [15].

However, Jaccard similarity does not agree with set containment, and the disagreement is exasperated when the sizes of the sets being compared differ significantly. The most recent approach to MinHash-based indexing that supports set containment is *Asymmetric Minwise Hashing* [24]. This approach "pads" the domains with fresh values so they end up with equal sizes. Shrivastava and Li show that near neighbors based on Jaccard similarity and padded MinHash signatures converge to near neighbors with set containment [24]. We experimentally show here that padding used on Open Data may decrease recall. As we will show in Section 6, with a finite number of hash functions, increased skewness in the domain sizes may lead to a significant decrease in recall. LSH, Asymmetric Minwise Hashing, and other related work is presented in greater detail in Section 4.

1.2 Contributions

The main contributions of this paper are as follows.

- We define the domain search problem using set containment as the relevance measure so that we can find domains that maximally contain a query domain. We use domain search to find joinable tables.
- We present a new indexing structure, the *LSH Ensemble*, to address the domain search problem for datasets found in Open Data and Web data repositories at an Internet scale. LSH Ensemble is an efficient and scalable solution to the domain search problem.
- We propose a data partitioning scheme that makes LSH Ensemble accurate over domains whose sizes are skewed (following a power-law distribution).
- We present a cost model for precision and recall of any partitioning, enabling us to formulate data partitioning for LSH Ensemble as an optimization problem.

- We prove the existence of an optimal partitioning for any data distribution. In particular, for power-law distributions (which we have empirically observed in Canadian Open Data and the WDC Web Tables Corpus 2015), we show that the optimal partitioning can be approximated using equi-depth, allowing for efficient implementation.
- Experimentally, we evaluate our implementation of LSH Ensemble using both the Canadian Open Data repository and WDC Web Tables Corpus 2015. We show that LSH Ensemble scales to hundreds of millions of domains with each domain having up to millions of distinct values, while consistently sustaining query response time of a few seconds.
- We compare our approach against the state-of-the-art MinHash LSH [15] and Asymmetric Minwise Hashing [24]. Compared to these alternatives, our approach significantly improves precision while maintaining high recall.

We believe that LSH Ensemble is an essential tool that will enable data scientists to find new, relevant Open Data and Web Data for their data analysis projects. By finding domains that maximally contain a query dataset, our approach can help scientists to find datasets that best augment their own data (by maximally joining with their data).

2. PROBLEM DEFINITION

By *domain*, we mean a set of data values. A data set R can be characterized by a collection of *domains*. When R is a relational table, the domains are given by the projections $\pi_i(R)$ on each of the attributes of R. We write $\text{dom}(R)$ to denote the set of all domains of R.

In Section 1.1, our motivation is, given a source data set R, we want to discover data sets which are joinable with R. We assume that, given a data set W, R and W are highly joinable if there is a pair of domains $X \in \text{dom}(R)$ and $Y \in \text{dom}(W)$ such that X is mostly contained by Y. Namely $|X \cap Y|/|X|$ is large.

Definition 1 (Set containment). *The set containment of X in Y is defined as:*

$$t(X, Y) = \frac{|X \cap Y|}{|X|} \tag{1}$$

We formalize the *domain search problem* as a R-nearest neighbor problem.

Definition 2 (Domain Search). *Given a collection of domains $\mathcal{D}$, a query domain Q, and a threshold $t^* \in [0, 1]$ on the set containment score, find a set of relevant domains from $\mathcal{D}$ defined as*

$$\{X : t(Q, X) \geq t^*, X \in \mathcal{D}\} \tag{2}$$

In addition to set containment, one can also quantify the relevance of two domains X and Y using *Jaccard similarity* of the respective domains [6]. The Jaccard similarity between domains X and Y is defined as:

$$s(X, Y) = \frac{|X \cap Y|}{|X \cup Y|} \tag{3}$$

While Jaccard similarity is the right measure for many applications, we argue that set containment, which is asymmetric, is better suited for domain search. Consider for example a query domain `Q`, and two domains, `Provinces` and `Locations`, as follows:

```
        Q = {Ontario, Toronto}
Provinces = {Alberta, Ontario, Manitoba}
Locations = {Illinois, Chicago, New York City
             New York, Nova Scotia, Halifax,
             California, San Francisco, Seattle,
             Washington, Ontario, Toronto}
```

Is `Q` closer to `Provinces` or to `Locations`? The Jaccard similarity of `Q` and `Provinces` is 0.25, while that of `Q` and `Locations` is 0.083. According to these scores, domain `Q` is more relevant to domain `Provinces` than `Locations`, which is a bit counter intuitive if we look at the data values. On the other hand, the set containment of `Q` and `Provinces` is 0.5 while that of `Q` and `Locations` is 1.0. According to these set containment scores, domain `Q` is judged to be more relevant to `Locations` than `Provinces`. This example shows that Jaccard similarity favors domains with small size (`Provinces`). Such bias is undesirable for the purpose of discovery of joinable Web data. Set containment, on the contrary, is agnostic to the difference in the sizes of the domains.

The domain search problem distinguishes itself from the traditional Web keyword search in that the query itself is an entire domain with arbitrarily large cardinality. It would be impractical to treat each value in the query domain as a keyword. Thus, we are interested in a search algorithm with approximately constant time query complexity. In addition, as discussed in the introduction, the domain containment search problem has the following constraints.

- $|\mathcal{D}|$ is on the order of hundreds millions.
- $\{|X| : X \in \mathcal{D}\}$ is very skewed.
- $|Q|$ can be arbitrarily large.

Given such constraints, we look for an **approximate** solution to the domain containment search problem, with the following properties.

- *Efficient indexing*: The space complexity of the index structure should be *almost constant* (growing very sublinearly) with respect to the domain sizes, and linear with respect to the number of domains. An index structure can be built in a single-pass over the data values of the domains.
- *Efficient search*: The search time complexity should be constant with respect to the query domain size and sublinear with respect to the number of domains in $\mathcal{D}$.

An alternative way of formulating the problem is by defining it as searching for the top-k relevant domains. Since we are interested in discovering domains that can be used in value-based joins, search by containment threshold is more natural. The top-k relevant domains are not guaranteed to have a sufficient level of overlap with the given query domain. We remark that the two formulations (top-k versus threshold) are closely related and complementary [14].

3. PRELIMINARIES

Our solution to the domain containment search problem is built on the foundation of Minwise Hashing [6] and LSH [15]. We present a brief overview of these two techniques.

3.1 Minwise Hashing

Broder [6] proposed a technique for estimating the Jaccard similarity between domains of any sizes. In this technique, a domain is converted into a *MinHash signature* using a set of

minwise hash functions. For each minwise hash function, its hash value is obtained by using an independently generated hash function that maps all domain values to integer hash values and returns the minimum hash value it observed.

Let $h_{\min}$ be one such hash function and the minimum hash value of a domain X be $h_{\min}(X)$ and Y be $h_{\min}(Y)$. Broder showed that the probability of the two minimum hash values being equal is the Jaccard similarity of X and Y:

$$P(h_{\min}(X) = h_{\min}(Y)) = s(X, Y) \quad (4)$$

Thus, given the signatures of X and Y, we can obtain an unbiased estimate of the Jaccard similarity by counting the number of *collisions* in the corresponding minimum hash values and divide that by the total number of hash values in a single signature.

3.2 Locality Sensitive Hashing

The Locality Sensitive Hashing (LSH) index was developed for general approximate nearest neighbor search problem in high-dimensional spaces [15]. LSH can be applied to the approximate R-near neighbor search problem [2]. An LSH index requires a family of LSH functions. An LSH function is a hash function whose collision probability is high for inputs that are close, and low for inputs that are far-apart. The distance measure must be symmetric. A formal definition of LSH functions can be found in the work of Indyk and Motwani [15].

The minwise hash function $h_{\min}$ belongs to the family of LSH functions for Jaccard distance, as the probability of collision is equal to the Jaccard similarity. For simplicity, we call the LSH index for Jaccard similarity *MinHash LSH*.

Next we explain how to build a MinHash LSH index. Assume we have a collection of MinHash signatures of domains generated using the same set of minwise hash functions. The LSH divides each signature into b "bands" of size r. For the i-th band, a function $H_i = (h_{\min,1}, h_{\min,2}, \ldots, h_{\min,r})$ is defined, which outputs the concatenation of the minimum hash values in that band, namely, the values $h_{\min,1}$ to $h_{\min,r}$. The function H_i maps a band in a signature to a bucket, so that signatures that agree on band i are mapped to the same bucket.

Given the signature of a query domain, LSH maps the signature to buckets using the same functions $H_1, \ldots, H_b$. The domains whose signatures map to at least one bucket where the query signature is mapped are the *candidates*. These candidates are returned as the query result. Hence, the search time complexity of the LSH only depends on the number of minwise hash functions (or the signature size) and is sub-linear with respect to the number of domains indexed.

The probability of a domain being a candidate is a function of the Jaccard similarity s between the query and the domain. Given the parameters b and r, the function is:

$$P(s|b, r) = 1 - (1 - s^r)^b \quad (5)$$

Given a threshold on Jaccard similarity, we want the domains that meet the threshold to have high probability of becoming candidates, while those do not meet the threshold to have low probability. This can be achieved by adjusting the parameters b and r.

4. RELATED WORK

In this section, we provide a brief survey of previous work related to domain search from different perspectives.

IR Indexes for Keyword Search. Indexes for searching structured datasets given a user-specified keyword query have been extensively studied [9, 13, 16, 22]. This work is similar to domain search as search results should contain the keywords in the query. However, in domain search, the query itself is a domain containing (possibly) large number of data values. To use keyword search to solve domain search, we could generate a keyword for each element in the domain. Such a solution can only be used for small domains. We need to handle very large domains containing millions of distinct values. In our solution, the index search time complexity is constant time with respect to the query domain size. In addition, the relevance measures in keyword search engines, such as the tf-idf score, consider the frequency of each keyword in all documents. This is different from the relevance measure in domain search, which considers the degree of overlap between the query and the indexed domain.

Other LSH Indexes. Various LSH indexes have been developed for distance metrics other than Jaccard, including Euclidean [11], Hamming [15], and Cosine distance [8], among others. These approaches are widely used in areas such as image search and document deduplication. Many such LSH variations require the data to be vectors with fixed dimensions. While it is possible to convert domains into binary vectors, where each unique data value is a dimension, binary vector representation is not practical for domain search at Internet-scale. These indices require prior knowledge of the values in the query domain, hence, they would need to be rebuilt whenever a domain with unseen values is inserted. Thus, LSH indexes, such as MinHash LSH, that do not require a fixed set of domain values are better suited for Internet-scale domain search.

SimHash is an LSH index for Cosine distance [8], and it is applicable to domain search, since it does not prescribe a fixed set of values for all domains. However, it has been shown that MinHash LSH is more computationally efficient than SimHash [23]. Hence, we use MinHash LSH in our experimental evaluation (Section 6).

Asymmetric Minwise Hashing. Asymmetric Minwise Hashing by Shrivastava and Li is the state-of-the-art technique for containment search over documents [24]. Asymmetric Minwise Hashing applies a pre-processing step called the asymmetric transformation, that injects fresh padding values into the domains to be indexed.[1] This makes all domains in the index have the same size as the largest domain. They show that Jaccard similarity between an untransformed signature (for the query) and a transformed signature (for index domain) is monotonic with respect to set containment. Thus, MinHash LSH can be used to index the transformed domains, such that the domains with higher set containment scores will have higher probabilities of becoming candidates. While this technique is very useful for containment search over documents such as emails and news articles, we observed that the asymmetric transformation reduces recall when the domain size distribution is very skewed. Since Asymmetric Minwise Hashing is the only technique that directly addresses containment search, we include it in our experimental comparison in Section 6.

Schema and Ontology Matching Techniques. Examples of unsupervised instance-based schema matching tech-

[1]For efficiency, the padding is done to the MinHash signatures of the domains rather than to the domains themselves.

Table 2: Symbols Used in Analysis

Symbol	Description
X, Q	Indexed domain, Query domain
x, q	Domain size of X, Domain size of Q
b, r	Number of bands in LSH, Number of hash functions in each band
$s(Q,X)$, $t(Q,X)$	Jaccard similarity and containment score between sets Q and X
s^*, t^*	Jaccard similarity and containment thresholds
$\hat{s}_{x,q}(t)$, $\hat{t}_{x,q}(s)$	functions that convert $t(Q,X)$ to $s(Q,X)$ and vice versa, given the domain sizes $x=\|X\|$ and $q=\|Q\|$
FP, FN	False positive & negative probabilities
N^{FP}	Count of false positive domains
l, u	Lower, Upper bound of a partition
m	Num. hash functions in MinHash
n	Num. of partitions in LSH Ensemble

niques are COMA++ [3] and DUMAS [5]. These approaches rely on overlap or similarity of instance values. In order to make instance-based matchers scalable, Duan et al. [12] use MinHash LSH and random projection (simhash) to do type matching based on Jaccard and Cosine similarity. The goal of LSH Ensemble is to match a query domain against extremely large number of domains with domain sizes varying from a few to millions. LSH Ensemble can be applied in large-scale schema and ontology matching, but to the best of our knowledge, no schema matchers scale to millions of attributes.

Methods for Table Search. Table search is the problem of finding join candidates in a repository of relational tables. Semantic-based table search approaches enrich tables with ontology-based semantic annotations [25, 20]. Semantic-based table search is complementary to our approach which uses set containment. Some table search approaches rely on table context and metadata in addition to table content to do search [19, 26]. For instance, InfoGather uses a matching measure consisting of features such as the similarity of text around tables in Web pages, the similarity of each table to its own context, the similarity of the URL of the web page containing each table, and the number of overlapping tuples in tables [26]. Open datasets often lack text context, description and any schema related information. Our approach relies solely on the values of domains in tackling the domain search problem. LSH Ensemble can be used as a component in table search engines as it searches for highly overlapping attributes to a query domain in Web scale. Note that other table search approaches are not able to do containment search over hundreds of millions of domains as in LSH Ensemble.

5. LSH ENSEMBLE

In this section, we describe our contribution, *LSH Ensemble*. In our approach, the domains are indexed in *two stages*. In the first stage, the domains are partitioned into disjoint partitions based on the domain cardinality. In the second stage, we construct a MinHash LSH index for *each* partition. Each LSH index is dynamically tuned with its specific Jaccard similarity threshold. In Section 5.1 to Section 5.3, we relate set containment to Jaccard similarity, and present a cost model that describes the resulting false positive rate if a conservative Jaccard similarity threshold is used to perform set containment search.[2] Our main contribution is presented in Section 5.4. We present a partitioning strategy that optimally minimizes the false positive rate if an ideal Jaccard similarity filter is used in each partition. We use dynamic LSH indexes to implement the Jaccard similarity filter, as discussed in Section 5.5. We optimally configure the parameters so that the inherent false positive and false negative errors associated with the LSH indexes are minimized.

Table 2 is a summary of the symbols used in the paper.

5.1 Similarity Filtering for Containment

For the containment search problem, we are given a desired minimal containment as a threshold t^*. On the other hand, LSH can only be tuned given a Jaccard similarity threshold s^*. Thus, in order to use LSH for containment search, the containment threshold needs to be converted to a Jaccard similarity threshold.

Consider a domain X with domain size $x=|X|$ and query Q with domain size $q=|Q|$. We can convert Jaccard similarity and containment back and forth by the inclusion-exclusion principle. If $t=t(Q,X)$, we can compute the corresponding Jaccard similarity $s=s(Q,X)$ given the domain sizes x and q. The transformations are given as follows.

$$\hat{s}_{x,q}(t)=\frac{t}{\frac{x}{q}+1-t},\ \hat{t}_{x,q}(s)=\frac{(\frac{x}{q}+1)s}{1+s} \tag{6}$$

Notice the transformation $t^*\mapsto s^*$ depends on the domain size x, which is typically not a constant, so we need to approximate x. We choose to do so in a way that ensures the transformation to s^* does not introduce any new false negatives over using t^*. Suppose that X is from a *partitioned* set of domains with sizes in the interval of $[l,u)$, where l and u are the lower and upper domain size bound of the partition. A conservative approximation can be made by using the upper bound u for x. This ensures that filtering by s^* will not result in any new false negatives.

We define a Jaccard similarity threshold using the upper bound u:

$$s^*=\hat{s}_{u,q}(t^*)=\frac{t^*}{\frac{u}{q}+1-t^*} \tag{7}$$

while the exact Jaccard similarity threshold is $\hat{s}_{x,q}(t^*)$. Because $u\geq x$ and $\hat{s}_{x,q}(t^*)$ decreases monotonically with respect to x, we know $s^*=\hat{s}_{u,q}(t^*)\leq\hat{s}_{x,q}(t^*)$. Thus, by using this approximation for s^*, we avoid false negatives introduced by the approximation.

The search procedure is described in Algorithm 1. The function **approx**($|Q|$) provides an estimation of the query domain size; this can be done in constant time using the MinHash signature of the query [10]. We remark that the choice of s^* will not yield any false negatives, provided that `Similarity-Search` perfectly filters away all domains with

[2]By conservative, we mean that we choose a Jaccard similarity threshold that guarantees zero false negatives.

Algorithm 1: Domain Search with Set Containment

Containment-Search($\mathbf{I}(\mathcal{D})$, MinHash(Q), t^*)
Input: $\mathbf{I}(\mathcal{D})$: an index of a collection of domains
MinHash(Q): the MinHash signature of a query domain
t^*: containment threshold

1 let $l \leftarrow \min\{|X| : X \in \mathcal{D}\}$, $u = \max\{|X| : X \in \mathcal{D}\}$
2 let $q \leftarrow \textbf{approx}(|Q|)$
3 let $s^* = \hat{s}_{u,q}(t^*)$
4 let $\mathcal{D}_{\text{candidates}} = \texttt{Similarity-Search}(\mathbf{I}(\mathcal{D}), Q, s^*)$
5 **return** $\mathcal{D}_{\text{candidates}}$

Jaccard similarity less than s^*. Of course, this may not be the case, but our conservative choice of s^* using u guarantees no new false negatives are introduced. We will describe the index $\mathbf{I}(\mathcal{D})$ and the `Similarity-Search` shortly.

An LSH Ensemble partitions the set of *all* domains in $\mathcal{D}$ by their domain size into disjoint intervals. Let $[l_i, u_i)$ be the lower and upper bounds of the i-th partition, and $\mathcal{D}_i = \{X \in \mathcal{D} : l_i \leq |X| < u_i\}$, where $i = 1 \cdots n$. With n partitions, we can search each partition and take the union of the individual query answers. Algorithm 1 provides the search procedure for a *single* partition.

$$\begin{aligned} &\texttt{Partitioned-Containment-Search}(\{\mathbf{I}(\mathcal{D})\}, \text{MinHash}(Q), t^*) \\ = &\bigcup_i \texttt{Containment-Search}(\mathbf{I}(\mathcal{D}_i), \text{MinHash}(Q), t^*) \end{aligned}$$

The `Partitioned-Containment-Search` can be evaluated concurrently where `Containment-Search` can be evaluated in parallel, and their results are unioned.

5.2 A Cost Model for Containment Search

Let the time complexities of `Containment-Search` and `Similarity-Search` be denoted by $T_{\text{containment}}$ and $T_{\text{similarity}}$ respectively.

The complexity of Algorithm 1 is given by:

$$T_{\text{containment}} = T_{\text{similarity}} + \Theta(\text{correct result}) + \Theta(N^{\text{FP}}_{l,u}) \quad (8)$$

The last two terms are the cost of processing the query result. The value $N^{\text{FP}}_{l,u}$ is the number of false positives from `Similarity-Search` in the partition $[l, u)$. In Equation 8, we only want to minimize the term $\Theta(N^{\text{FP}}_{l,u})$ to reduce the overall cost.

To minimize the time complexity of the parallel evaluation of `Partitioned-Containment-Search`, we wish to minimize the following cost function by designing the partitioning intervals $[l_i, u_i)$:

$$\text{cost} = \max_{1 \leq i \leq n} N^{\text{FP}}_{l_i,u_i} \quad (9)$$

Before we can use this cost model, we must be able estimate the number of false positive in a partition. This cost model and FP estimation will allow us to compute the expected cost of a particular partition and develop an optimal partitioning.

5.3 Estimation of False Positives

By using Jaccard similarity filter with threshold s^* instead of the containment threshold t^*, we incur false positives in the search result of `Similarity-Search` even if no error is introduced by using MinHash signatures and $\mathbf{I}(\mathcal{D})$.

Consider a domain X with size x in some partition $[l, u)$. Domain X would be a false positive if it meets the passing condition of the approximated Jaccard similarity threshold, but actually fails the containment threshold.

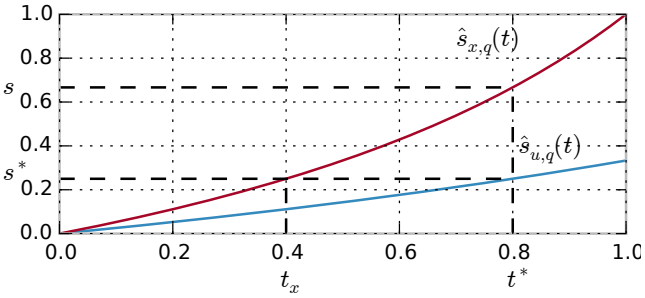

Figure 2: The relationships among t_x, t^*, and s^*

Let the containment of the domain be t, and the Jaccard similarity be s. The domain is a false positive if $t < t^*$ but $s > s^*$. `Similarity-Search` filters by s^*, so it is filtering X according to an effective containment threshold given by $\hat{t}_{x,q}(s^*)$. Define $t_x = \hat{t}_{x,q}(s^*)$. Thus, a domain is a false positive if its containment t falls in the interval of $[t_x, t^*]$. Namely, its true containment score is below the query threshold t^*, but above the effective threshold t_x. The relationship between t_x and t^* is illustrated by Figure 2, which plots the Jaccard similarities $\hat{s}_{x,q}(t)$ and $\hat{s}_{u,q}(t)$ as functions of containment t, with $u = 3$, $x = 1$ and $q = 1$.

PROPOSITION 1. *The effective containment threshold for X is related to the query containment threshold by the following relation.*

$$t_x = \frac{(x+q)t^*}{u+q} \quad (10)$$

Given no prior knowledge about the domain X, we assume its containment is uniformly distributed in the interval $[0, 1]$. Thus, we can estimate the probability of a true negative being falsely identified as a candidate.

$$P(X \text{ is FP}) = (t^* - t_x)/t^* \quad (11)$$

Let $N_{l,u}$ be the expected total number of domains with sizes in the interval $[l, u)$. The expected number of false positives produced by the threshold is given by

$$N^{\text{FP}}_{l,u} = \sum\{P(X \text{ is FP}) : X \in \mathcal{D}, l \leq |X| < u\} \quad (12)$$

where $N_{l,u}$ is the total number of domains that belong to the partition $[l, u)$.

If we further assume that, within a partition, the distribution of x is uniform, then we can evaluate Equation 12 further to obtain the following result.

PROPOSITION 2. *Assuming a uniform distribution of domain sizes in the interval $[l, u)$, the upper bound of the number of candidate domains which are false positives is given by*

$$N^{\text{FP}}_{l,u} \leq N_{l,u} \cdot \frac{u - l + 1}{2u} \quad (13)$$

PROOF. (Outline) We need to cover several cases: (1) $t^*q \leq l$, (2) $t_l q \leq l < t^*q$ and $t^*q \leq u$, (3) $l < t_l q$ and $t^*q \leq u$, (4) $l < t_l q$ and $t_u q \leq u < t^*q$, and finally (5)

$u < t_u q$. For the first case $t^* q \leq l$, we have the probability of a domain with size x being a false positive being $(t^* - t_x)/t$.

$$N^{\text{FP}}_{l,u} = N_{l,u} \sum_{x=l}^{u-1} \frac{t^* - tx}{t^*} \frac{1}{u-l} dx \tag{14}$$

$$= N_{l,u} \cdot \frac{u-l+1}{2(u+q)} \leq N_{l,u} \cdot \frac{u-l+1}{2u} \tag{15}$$

If $u \gg q$, then the upper bound is a tight upper bound. The other cases (2-5) are proven similarly. □

To summarize, by assuming uniform distribution of domain size within an interval, we can compute an upper bound of the number of false positives in the interval given only its boundaries $[l, u)$ and its number of domains $N_{l,u}$.

5.4 Optimal Partitioning

The cost function defined in Equation 9 and the estimate of false positive candidate domains in Equation 13 allow us to compute the expected cost of a particular partition. In this section, we provide a concrete construction of a minimal cost partitioning.

We denote a partitioning of the domains in $\mathcal{D}$ by $\Pi = \langle [l_i, u_i) \rangle_{i=1}^n$ where $l_{i+1} = u_i$. Let $N^{\text{FP}}_i = N^{\text{FP}}_{l_i,u_i}$. An optimal partitioning is then one that minimizes the number of false positives over all possible partitions.

Definition 3 (The Optimal Partitioning Problem). *A partitioning Π^* is an optimal partitioning if*

$$\Pi^* = \arg\min_{\Pi} \max_i N^{\text{FP}}_i$$

We now show that there is an optimal partitioning that has equal number of false positives in each partition.

Theorem 1 (Equi-FP optimal partitioning). *There exists an optimal partitioning Π^* such that $N^{\text{FP}}_i = N^{\text{FP}}_j$ for all i, j.*

Proof. (Outline) Recall that the cost function is $\text{cost}(\Pi) = \max_i N^{\text{FP}}_i$. We can show that N^{FP}_i is monotonic with respect to $[l_i, u_i)$. For simplicity, suppose we only have two partitions, $n = 2$. Suppose we have an optimal partitioning Π_1 such that $N^{\text{FP}}_1 \neq N^{\text{FP}}_2$. Without loss of generality, we assume that $N^{\text{FP}}_1 < N^{\text{FP}}_2$. By the monotonicity of N^{FP} w.r.t. the interval, we can increase the value of u_1 such that N^{FP}_1 is increased and N^{FP}_2 is decreased until $N^{\text{FP}}_1 = N^{\text{FP}}_2$. The new partitioning Π_2 obtained by the adjustment of u_1 is such that $\text{cost}(\Pi_2) \leq \text{cost}(\Pi_1)$, so it is also optimal. This proof generalizes to arbitrary n. □

As a result of Theorem 1, we can construct an equi-N^{FP}_i partitioning Π as a way to optimally partition the domains. Unfortunately, this means that Π is query dependent because $N^{\text{FP}}_i = N^{\text{FP}}_i(q)$. We cannot afford to repartition $\mathcal{D}$ for each query. Fortunately, with a reasonable assumption, there exists a *query independent* way of partitioning $\mathcal{D}$ near-optimally. We assume that $\max\{|X| : X \in \mathcal{D}\} \gg q$. Namely, $\mathcal{D}$ contains *large* domains relative to the query. An index will most often be used with queries that are much smaller than the maximum domain size.

Let M_i be the upper bound on N^{FP}_i following Proposition 2:

$$M_i = N_{l_i,u_i} \cdot \frac{u_i - l_i + 1}{2u_i} \tag{16}$$

Let Π^* be an optimal equi-N^{FP}_i partitioning whose existence is guaranteed by Theorem 1. Under the large domain assumption, we can see that

$$\text{cost}(\Pi^*) = N^{\text{FP}}_n \approx M_n \tag{17}$$

which is query independent. This suggests that we can *approximate* the optimal partitioning Π^* by an equi-M_i partitioning of $\mathcal{D}$. Furthermore, we show that for power-law distribution, equi-M_i partitioning is the same as equi-N_{l_i,u_i}, or *equi-depth*.

Theorem 2 (Partitioning and power-law). *Suppose $\mathcal{S} = \{|X| : X \in \mathcal{D}\}$ follows a power-law distribution with frequency function $f(x) = Cx^{-\alpha}$ where $\alpha > 1$. The equi-N_{l_i,u_i} partitioning*

$$\Pi = \langle [l_i, u_i) \rangle_{i=1}^n, \; N_{l_i,u_i} = |\mathcal{S}|/n, \; \forall i = 1 \ldots n \tag{18}$$

is an equi-M_i partitioning.

Proof. We use the (tight) bound on number of false positives defined by Equation 16. Using the property of equi-N_{l_i,u_i} partitioning, we have

$$N_{l_i,u_i} = \int_{l_i}^{u_i} Cx^{-\alpha} dx = |\mathcal{S}|/n, \; \forall i = 1 \ldots n \tag{19}$$

Solving the equation above we obtain a relationship between l_i and u_i:

$$\frac{l_i}{u_i} = \left| 1 - \frac{|\mathcal{S}|(\alpha-1) l_i^{\alpha-1}}{nC} \right|^{-(\alpha-1)} = g(l_i), \; \forall i = 1 \ldots n \tag{20}$$

With the expressions for u_i and l_i, we get

$$\frac{u_i - l_i + 1}{2u_i} = \frac{1}{2} - \frac{g(l_i)}{2} + \frac{1}{2u_i} \approx \frac{1}{2}$$

Since $g(l_i)$ and $\frac{1}{2u_i}$ approach 0 fast with increasing l_i and u_i respectively, and are likely very small comparing to $\frac{1}{2}$ in practice when $|S|$ is in the millions and $2u_i \gg 1$, we can discard them, and obtain a constant for all partitions. Given that both terms in Equation 16 are constants, the equi-N_{l_i,u_i} partitioning is an equi-M_i partitioning. □

Theorem 2 provides a fast method for approximating the optimal partitioning. For a collection of domains, we can create a desired number of partitions such that all contain the same number of domains. Based on Theorem 2, if the domain size distribution follows a power-law, this is an equi-M_i partitioning and hence, a good approximation to equi-N^{FP}_i partitioning, which is optimal.

5.5 Containment Search Using Dynamic LSH

In Algorithm 1, we assume the existence of a search algorithm based on a Jaccard similarity threshold for *each* partition $\mathcal{D}_i$. We choose to index the domains in $\mathcal{D}_i$ using a MinHash LSH index with parameters (b, r) where b is the number of bands used by the LSH index, and r is the number of hash values in each band.

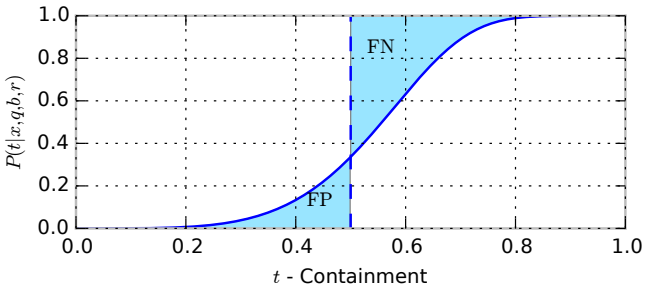

Figure 3: $P(t|x,q,b,r)$ - probability of becoming candidate with respect to containment, given $x = 10$, $q = 5$, $b = 256$, and $r = 4$; containment threshold $t^* = 0.5$ (dashed line)

Traditionally, MinHash LSH has a fixed (b, r) configuration, and thus has a static Jaccard similarity threshold given by the approximation:

$$s^* \approx (1/b)^{(1/r)} \tag{21}$$

One can observe that MinHash LSH only approximates `Similarity-Search`, and therefore will introduce false positives, in addition to N^{FP} in each partition, and also false negatives. In this section, we describe our method of selecting (b, r) so that the additional error due to the approximation nature of MinHash LSH is minimized. Since s^* is query dependent, we choose to use a dynamic LSH index, such as the LSH Forest [4], so we can vary (b, r) for each query. LSH Forest uses a prefix tree to store the r hash values in each band (see Section 3.2), thus the effective value of r can be changed at query time by choosing the maximum depth to traverse to in each prefix tree. The parameter b can be varied by simply choosing the number of prefix trees to visit.

We now describe how the parameters (b, r) are selected so that the resulting selectivity agrees with s^*. Using the relationship between Jaccard similarity s and containment t, we express the probability of a domain X becoming a candidate in terms of $t = t(Q, X)$, $x = |X|$, and $q = |Q|$ instead of $s = s(Q, X)$.

$$P(t|x,q,b,r) = 1 - (1 - s^r)^b = 1 - \left(1 - \left(\frac{t}{\frac{x}{q} + 1 - t}\right)^r\right)^b \tag{22}$$

Figure 3 plots the probability (for $t^* = 0.5$), along with the areas corresponding to the false positive (FP) and false negative probabilities (FN) induced by the MinHash LSH approximation. It is important to notice that the false positives here are introduced by the use of `Similarity-Search` with MinHash LSH, which is different from the false positive introduced by the transformation of the containment threshold to a Jaccard similarity threshold in Section 5.3.

Note that t cannot exceed the size ratio x/q. We can express the probability of X being a false positive, FP, or a false negative, FN, in terms of t^* and x/q.

$$\text{FP}(x,q,t^*,b,r) = \begin{cases} \int_0^{t^*} P(t|x,q,b,r)dt & \frac{x}{q} \geq t^* \\ \int_0^{\frac{x}{q}} P(t|x,q,b,r)dt & \frac{x}{q} < t^* \end{cases} \tag{23}$$

$$\text{FN}(x,q,t^*,b,r) = \begin{cases} \int_{t^*}^1 1 - P(t|x,q,b,r)dt & \frac{x}{q} \geq 1 \\ \int_{t^*}^{\frac{x}{q}} 1 - P(t|x,q,b,r)dt & t^* \leq \frac{x}{q} < 1 \\ 0 & \frac{x}{q} < t^* \end{cases} \tag{24}$$

The optimization objective function for tuning LSH is given as:

$$\arg\min_{b,r} (\text{FN} + \text{FP})(x,q,t^*,b,r), \text{such that } 0 < br \leq m \tag{25}$$

where m is the number of minwise hash functions. Since x is not constant within a partition, we cannot use this objective function for tuning the LSH of the partition. As described in Section 5.1, for each partition i, we used u_i to approximate $x \in [l_i, u_i]$, and the alternative objective function for tuning we used is:

$$\arg\min_{b,r} (\text{FN}+\text{FP})(u_i,q,t^*,b,r), \text{such that } 0 < br \leq m \tag{26}$$

For some particular value of x, the b and r found using the alternative objective function would be more optimal if u_i were closer to x. Formally, for some $\epsilon > 0$, there exists $\delta > 0$ such that when $u_i - x < \delta$,

$$(\text{FP}+\text{FN})(x,q,t^*,b_p,r_p) - (\text{FP}+\text{FN})(x,q,t^*,b_{\text{opt}},r_{\text{opt}}) < \epsilon$$

where b_p and r_p are the parameters found using the alternative objective function, and b_{opt} and r_{opt} are the optimal parameters computed with the exact objective function.

The computation of (b, r) can be handled offline. Namely, we choose to pre-compute the FP and FN for different combinations of (b, r). At query time, the pre-computed FP and FN are used to optimize objective function in Equation 26. Given that b and r are positive integers and their product must be less than m, the computed values require minimal memory overhead.

Finally, we query each partition with the dynamically determined (b, r) using a dynamic LSH index as described by Bawa et al. [4].

In summary, the query evaluation of LSH Ensemble performs dynamic transformation of the containment threshold to a per-partition threshold of Jaccard similarity. Then we query each partition with different (b, r) to identify the candidate domains.

6. EXPERIMENTAL ANALYSIS

We evaluate the accuracy and performance of our LSH Ensemble technique, and compare against the traditional MinHash LSH [15] and the state-of-art Asymmetric Minwise Hashing [24] as baselines.

We have collected the relational data from the Canadian Open Data Repository (as of June 2015) which consists of 10,635 relations with 65,533 domains. The size of this data set allows us to compute the ground truth for any domain search queries, thus allowing us to reliably measure the accuracy of LSH Ensemble and the two baselines in Section 6.1. We evaluate the effectiveness of LSH Ensemble over dynamic data by showing the index is robust to changes in the distribution of the domain sizes (Section 6.2).

To test the performance LSH Ensemble at Internet scale, we used the entire English portion of the WDC Web Table Corpus 2015 [18] which consists of 51 million relations with 262 million domains (Section 6.3). The LSH ensemble for

Table 3: Experimental Variables

Variable	Range
Num. of Hash Functions in MinHash (m)	**256**
Containment Threshold (t^*)	0 - **0.5** - 1
Num. of Domains $\|\mathcal{D}\|$	65,533 - 262,893,406
Num. of Queries	**3,000**
Num. of Partitions (n)	8-**32**
Skewness	0.50 - **13.87**

the English Web Table corpus span over a cluster with 5 nodes, each with 8 Xeon processors and 64 GB memory.

We list the variables used in our experiments in Table 3, with default values in bold face.

6.1 Accuracy of LSH Ensemble

For our experimental evaluation, we use the set-overlap based definition of precision and recall. Let $\mathcal{D}$ be the set of domains in the index. Given a query domain Q and a containment threshold t^*, the ground truth set is defined as $T_{Q,t,\mathcal{D}} = \{X | t(Q,X) \geq t^*, X \in \mathcal{D}\}$. Let $A_{Q,t^*,\mathcal{D}}$ be the set of domains returned by a search algorithm. Precision and recall are defined as follows.

$$Precis. = \frac{|A_{Q,t^*,\mathcal{D}} \cap T_{Q,t^*,\mathcal{D}}|}{|A_{Q,t^*,\mathcal{D}}|}, Recall = \frac{|A_{Q,t^*,\mathcal{D}} \cap T_{Q,t^*,\mathcal{D}}|}{|T_{Q,t^*,\mathcal{D}}|} \tag{27}$$

We also used F_β score to evaluate the overall accuracy. We set β to 1 (equality weighted) and 0.5 (precision-biased). Since our algorithm is recall-biased, assigning more weight to precision gives a fairer evaluation of the overall accuracy.

$$\mathrm{F}_\beta = \frac{(1+\beta^2) \cdot Precis. \cdot Recall}{\beta^2 \cdot Precis. + Recall} \tag{28}$$

To evaluate accuracy, we use three main experiments over the CSV files from the Canadian Open Data repository for which we computed ground truth (exact containment scores). We did not use the WDC Web Table Corpus due to the extremely high time cost of obtaining the ground truth. We discarded domains with fewer than ten values and obtained 65,533 such domains which we indexed. Then, we sampled a subset of 3,000 domains and used them as queries. As we noted in Section 1, the domain size distribution of the Canadian Open Data roughly follows a power-law distribution (shown in Figure 1). Thus, we applied the equi-depth partitioning based on Theorem 2.

For a fair comparison, all the indexes including MinHash LSH and Asymmetric Minwise Hahsing are implemented to use the dynamic LSH algorithm for containment search described in Section 5.5, and the upper bound of domain sizes is used to convert containment threshold to Jaccard similarity threshold as described in Section 5.1.

Impact of Partitioning. Figure 4 shows the precision, recall, F-score and $\mathrm{F}_{0.5}$-score of MinHash LSH (Baseline), Asymmetric Minwise Hashing (Asym), and LSH Ensembles with different numbers of partitions (i.e., 8, 16, and 32). We report the average precision and recall for every containment threshold from 0.05 to 1.0 with a step size of 0.05. The equi-depth partitioning provides a clear improvement in precision over the baseline, verifying our theoretical analysis of partitioning. As the number of partitions grows, the precision increases for all containment thresholds; however the relative gain diminishes as the upper bound of each partition becomes a better approximation of the domain sizes of the partition. Recall decreases by about 0.02 each time the number of partitions doubles. The false negatives are introduced by the MinHash LSH in each partition, as described in Section 5.5. The index tuning becomes less conservative (regarding false negatives) as the upper bound of a partition approaches the actual domain sizes. There is a trade-off between partitioning and recall, however, since the approximation always remains conservative, the recall is not affected significantly. The F_β score results show that LSH Ensemble improved the overall accuracy over the baseline by as much as 25%. Since the threshold changes the optimizaiton constraints for index tuning (see Section 5.5) and the parameter space is integer, discontinuity may exist in the optimization objective. This leads to the sudden increase in precision at threshold 0.7.

Asymmetric Minwise Hashing achieved high precision comparable to that of LSH Ensemble, however, performed poorly in recall. Due to padding, some of the hash values in a MinHash signature are from the new padded values. For an indexed domain to become a candidate, its hash values from the original domain need to have collisions with the hash values in the signature of the query domain. Thus, with a finite number of hash functions, the probability of an indexed domain becoming a candidate, including domains that should be qualified, is lowered. This issue is not significant when the skew in domain sizes is small. However, when the skew is high, the amount of padding required becomes very large, making the probability of qualifying domains becoming candidates very low and sometimes nearly zero. This explains the low average recall over the skewed Open Data domains.[3] As the containment threshold increases, domains need to have a greater number of collisions in their hash values. Thus, the probability of finding qualifying domains drops to zero for high thresholds, resulting in a recall of zero. Around 80% of Asymmetric Minwise Hashing query results are empty for thresholds up to 0.7, more than 98% are empty for thresholds higher than 0.8, and 100% are empty for thresholds 0.95 and 1.0. We consider an empty result having precision equal to 1.0, however, we exclude such results when computing average precisions. For domain search, a high recall is a necessity. Thus, these results show that LSH Ensemble is a better choice for the domain search problem over skewed data.

Impact of Query Size. The equi-depth partitioning assumes the domain size of the query is much smaller than the maximum domain size of the partitioned index. We investigated whether in practice large query domain size influence the effectiveness of the partitioning technique. Figure 7 shows the precision and recall for queries with domain sizes in the smallest 10%, and Figure 6 shows the precision and recall for queries with domain sizes in the largest 10%. For the large queries, the precision is smaller, due to the assumption no longer holding. Still, the precision increases with more partitions, confirming our analysis that partitioning should always increase precision, and the recall stays high. The

[3]See the technical report for detailed analysis of Asymmetric Minwise Hashing: `http://arxiv.org/abs/1603.07410`

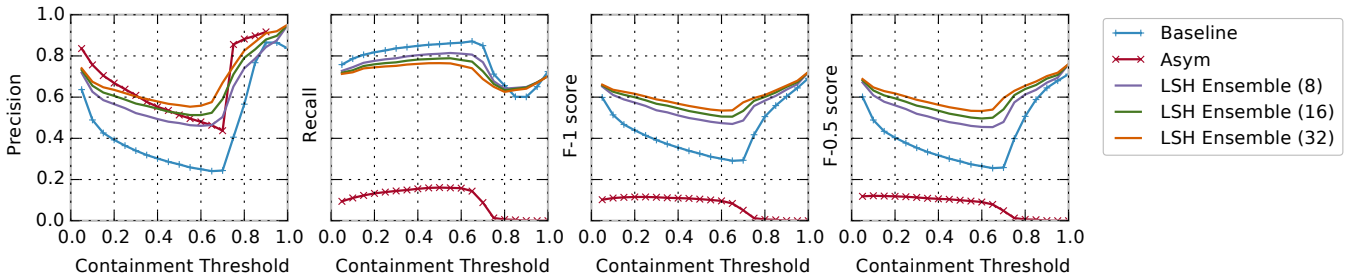

Figure 4: Accuracy versus Containment Threshold on Canadian Open Data Corpus

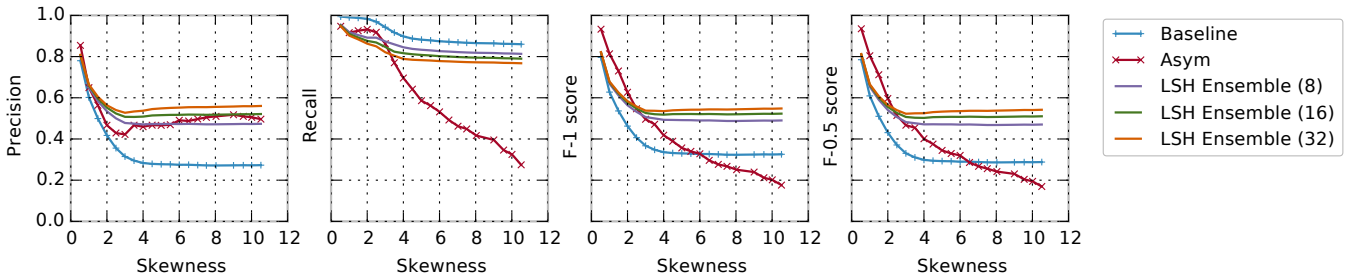

Figure 5: Accuracy versus Domain Size Skewness

result of the small queries is similar to the overall result in Figure 4, this is likely due to the fact that the domain sizes of the queries also follows a power-law distribution, so the queries contained mostly small domains.

Effect of Skewness on Accuracy. In order to investigate the effect of skew on accuracy. We created 20 subsets of the Canadian Open Data. The first contained a small (contiguous) interval of domain sizes. We then expanded the interval repeatedly to create 19 larger subsets. We measured the precision and recall of the MinHash LSH (Baseline), Asymmetric Minwise Hashing (Asym), and LSH Ensemble on each subset. Since the domain sizes approximately follow a power-law, the skewness increases as we use subsets with larger intervals of domain sizes. We compute the skewness of each subset using the following equation.

$$skewness = \frac{m_3}{m_2^{3/2}} \tag{29}$$

where m_2 and m_3 are the 2nd and 3rd moments of the distribution [17]. A higher positive skewness means there is more weight in the left tail of the distribution.

Figure 5 shows that as skewness increases, the precision of all indexes decreases, while a high level of recall is maintained except for Asymmetric Minwise Hashing. This is expected for MinHash LSH. Because it uses the upper bound of domain sizes in the conversion of containment to Jaccard similarity threshold (see Section 5.1), as the upper bound of domain sizes increases, the approximation becomes less precise, increasing the false positive rate. The same reason for the decrease in precision applies to LSH Ensemble as well, however, the issue is less severe. This is because of partitioning - the upper bound of each partition is a much better approximation than the overall upper bound. We can also observe that as the number of partitions goes up, the index is less affected by skew. As expected, Asymmetric Minwise Hashing achieved high recall when the skewness is small, but the recall decreases significantly as skewness increases. This is again due to the large amount of padding when the skewness is high.

We have also conducted experiments on evaluating the performance of using Asymmetric Minwise Hashing in conjunction with partitioning (and up to 32 partitions). Namely, we used Asymmetric Minwise Hashing instead of MinHash LSH in each partition. While there is a slight improvement in precision, we failed to observe any significant improvements in recall. This is due to the fact that, for a power-law distribution, some partitions still have sufficiently large difference between the largest and the smallest domain sizes, making Asymmetric Minwise Hashing unsuitable.

6.2 Dynamic Data

Web data will certainly be dynamic. While our index can easily accommodate the addition of new domains, if the new domains follow a different domain size distribution, our partitioning strategy may become less optimal (in particular, the partitions may start to have different sizes). Using domains from the Canadian Open Data, we simulated this scenario by creating partitionings that increasingly deviate away from equi-depth and move toward equi-width (meaning all partitions have the same interval size). We measured precision and recall of each partitioning, and computed the standard deviation among partition sizes.

Figure 8 shows that as the partitioning deviates away from equi-depth, the standard deviation increases. However, the precision remains almost the same until the standard deviation is increased beyond 5,556. This is more than 2.7 times the equi-depth partition size (of 2,047). This represents a huge shift in the domain size distributions before any degradation in accuracy is seen. So while the index may need to be rebuilt if the domain size distribution changes drastically, this would be a rare occurrence.

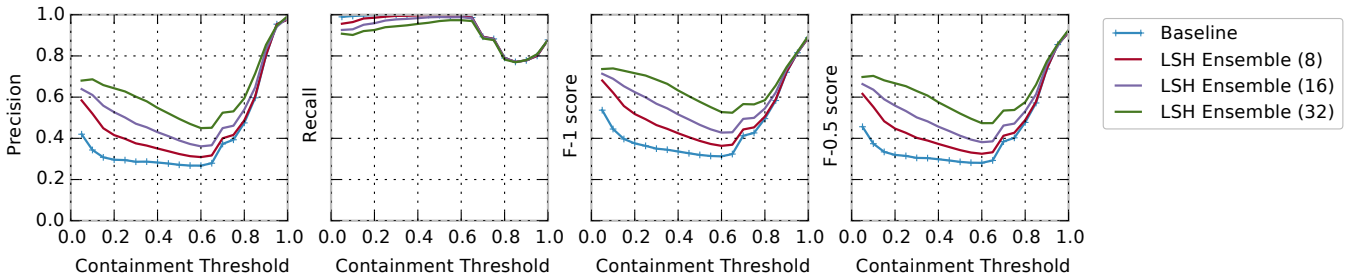

Figure 6: Accuracy of Queries with Large Domain Size (Highest 10%)

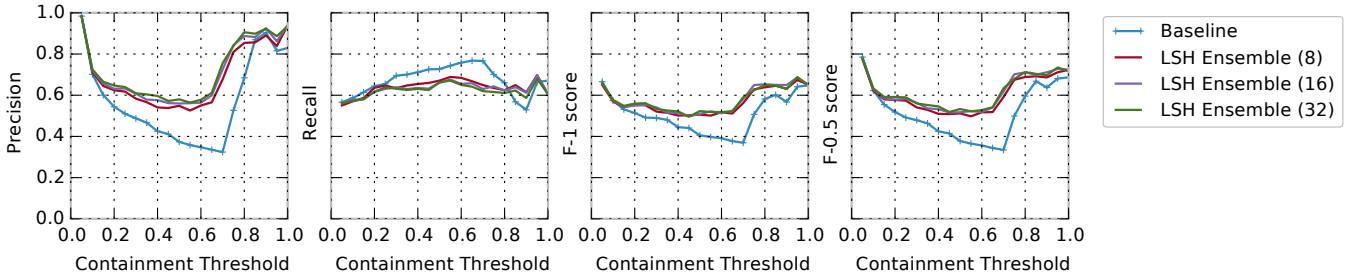

Figure 7: Accuracy of Queries with Small Domain Size (Smallest 10%)

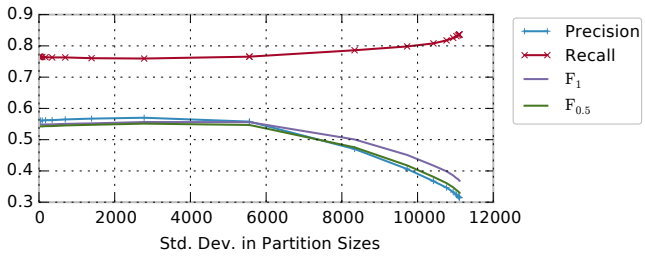

Figure 8: Accuracy vs. Std. Dev. of Partition Sizes

	Indexing (min)	Mean Query (sec)
Baseline	108.47	45.13
LSH Ensemble (8)	106.27	7.55
LSH Ensemble (16)	101.56	4.26
LSH Ensemble (32)	104.62	3.12

Table 4: Indexing and Query Cost of Baseline and LSH Ensemble on 263 Milion Domains

6.3 Efficiency of Indexing and Searching

For the performance experiments, we used the English relational subset of the WDC Web Table Corpus 2015, from which we extracted 262,893,406 domains. The domain size distribution closely resembles a power-law distribution, as shown in Figure 1. Thus, we used the equi-depth partitioning for LSH Ensemble as suggested by Theorem 2. We selected a random subset of 3,000 domains to use as test queries. Due to the massive number of domains, an index for all domains does not fit in the memory of a single machine. Thus, we divided the domains into 5 equal chunks on 5 machines, and then built an index for each chunk. A query client sends query requests to all indexes in parallel, and the results are unioned.

Table 4 compares the indexing and query cost of the basic MinHash LSH (baseline) with LSH Ensemble with different numbers of partitions. The indexing cost of all indexes are very similar. Since all partitions are indexed in parallel, the time to index does not increase with the number of partitions. On the other hand, the query cost of LSH Ensemble is significantly smaller than the MinHash LSH. In part of course, this is due to the parallelism, but in addition, we are seeing speed up because the partitioning improves precision. The index becomes more selective as the number of partitions increases. For 8 partitions, most of the speedup is parallelism, but for 16 partitions, the improved precision also improves the performance dramatically. Since all partitions are queried in parallel, the number of partitions does not contribute to query cost.

The first plot in Figure 9 shows the indexing cost with respect to the number of domains indexed. The indexing performance of LSH Ensemble scales linearly with respect to the number of domains. In addition, because of the parallel implementation, the number of partitions does not affect the indexing cost. The second plot in Figure 9 shows the query cost of LSH Ensemble. The query cost of LSH Ensemble increases with the number of domains in the index, because the number of candidates returned also increases (for a given threshold), and query processing cost includes the cost of outputting the candidates. On the other hand, the query cost increases much slower with more partitions. Again, this is because the precision is improved by partitioning, yielding a smaller selectivity.

7. CONCLUSION AND FUTURE WORK

We proposed the *domain search problem* where the goal is to find domains that maximally contain a query domain from Open Data repositories and other structured sources on the Web. We presented *LSH Ensemble*, a new index

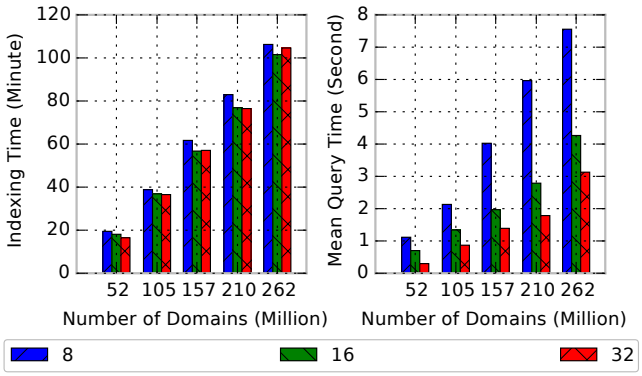

Figure 9: Indexing and Mean Query Cost

structure based on MinHash LSH, as a solution to the domain search problem using set containment. By means of partitioning, we show how we can efficiently perform set containment-based queries even on hundreds of millions of domains with highly skewed distribution of domain sizes. We constructed a cost model that describes the precision of LSH Ensemble with a given partition. We show that for any data distribution, there exists an optimal partitioning scheme that equalizes the false positives across the partitions. Furthermore, for datasets with a power-law distribution, the optimal partitioning can be approximated using equi-depth, amenable to efficient implementation.

We have conducted extensive evaluation of LSH Ensemble on the Canadian Open Data repository and the entire English relational WDC Web Table Corpus. Our solution is able to sustain query times of few seconds even at 262 million domains. Compared to other alternatives, LSH Ensemble consistently performs better in both accuracy and performance. In some cases, LSH Ensemble is $\sim$ 15 times faster than a MinHash LSH index for set containment queries.

8. ACKNOWLEDGMENT

This work was partially supported by NSERC.

9. REFERENCES

[1] P. Agrawal, A. Arasu, and R. Kaushik. On indexing error-tolerant set containment. In *SIGMOD*, pages 927–938, 2010.

[2] A. Andoni and P. Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. In *Communications of the ACM*, pages 117–122, 2008.

[3] D. Aumueller, H. H. Do, S. Massmann, and E. Rahm. Schema and ontology matching with COMA++. In *SIGMOD*, pages 906–908, 2005.

[4] M. Bawa, T. Condie, and P. Ganesan. LSH forest: Self-tuning indexes for similarity search. In *World Wide Web Conference*, pages 651–660, 2005.

[5] A. Bilke and F. Naumann. Schema matching using duplicates. In *ICDE*, pages 69–80, 2005.

[6] A. Broder. On the resemblance and containment of documents. In *Compression and Complexity of Sequences*, pages 21–28, 1997.

[7] M. J. Cafarella, A. Halevy, D. Z. Wang, E. Wu, and Y. Zhang. Webtables: Exploring the power of tables on the web. In *VLDB*, pages 538–549, 2008.

[8] M. S. Charikar. Similarity estimation techniques from rounding algorithms. In *STOC*, pages 380–388, 2002.

[9] J. Coffman and A. C. Weaver. An empirical performance evaluation of relational keyword search techniques. In *TKDE*, pages 30–42, 2014.

[10] E. Cohen and H. Kaplan. Summarizing data using bottom-k sketches. In *PODC*, pages 225–234, 2007.

[11] M. Datar, N. Immorlica, P. Indyk, and V. S. Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *SCG*, pages 253–262, 2004.

[12] S. Duan, A. Fokoue, O. Hassanzadeh, A. Kementsietsidis, K. Srinivas, and M. J. Ward. Instance-based matching of large ontologies using locality-sensitive hashing. In *ISWC*, pages 49–64, 2012.

[13] V. Hristidis and Y. Papakonstantinou. Discover: Keyword search in relational databases. In *VLDB*, pages 670–681, 2002.

[14] I. F. Ilyas, G. Beskales, and M. A. Soliman. A survey of top-k query processing techniques in relational database systems. *ACM Comput. Surv.*, 40(4):11:1–11:58, Oct. 2008.

[15] P. Indyk and R. Motwani. Approximate nearest neighbors: Towards removing the curse of dimensionality. In *STOC*, pages 604–613, 1998.

[16] M. Kargar, A. An, N. Cercone, P. Godfrey, J. Szlichta, and X. Yu. Meanks: meaningful keyword search in relational databases with complex schema. In *SIGMOD*, pages 905–908, 2014.

[17] S. Kokoska and D. Zwillinger. *CRC Standard Probability and Statistics Tables and Formulae*, chapter 2.2.24.1. Chapman & Hall/CRC, 2000.

[18] O. Lehmberg, D. Ritze, R. Meusel, and C. Bizer. A large public corpus of web tables containing time and context metadata. In *WWW*, 2016.

[19] O. Lehmberg, D. Ritze, P. Ristoski, R. Meusel, H. Paulheim, and C. Bizer. The mannheim search join engine. pages 159 – 166, 2015.

[20] G. Limaye, S. Sarawagi, and S. Chakrabarti. Annotating and searching web tables using entities, types and relationships. *Proc. VLDB Endow.*, 3(1-2):1338–1347, Sept. 2010.

[21] C. D. Manning, P. Raghavan, and H. Schütze. *Introduction to Information Retrieval*. Cambridge University Press, 2008.

[22] R. Pimplikar and S. Sarawagi. Answering table queries on the web using column keywords. In *VLDB*, volume 5, pages 908–919, 2012.

[23] A. Shrivastava and P. Li. In defense of minhash over simhash. In *AISTATS*, pages 886–894, 2014.

[24] A. Shrivastava and P. Li. Asymmetric minwise hashing for indexing binary inner products and set containment. In *WWW*, pages 981–991, 2015.

[25] P. Venetis, A. Halevy, J. Madhavan, M. Paşca, W. Shen, F. Wu, G. Miao, and C. Wu. Recovering semantics of tables on the web. *Proc. of VLDB Endowment (PVLDB)*, pages 528–538, 2011.

[26] M. Yakout, K. Ganjam, K. Chakrabarti, and S. Chaudhuri. Infogather: Entity augmentation and attribute discovery by holistic matching with web tables. In *SIGMOD*, pages 97–108, 2012.