Power-Law Based Estimation of Set Similarity Join Size *

Hongrae Lee
University of British Columbia
xguy@cs.ubc.ca

Raymond T. Ng
University of British Columbia
rng@cs.ubc.ca

Kyuseok Shim
Seoul National University
shim@ee.snu.ac.kr

ABSTRACT

We propose a novel technique for estimating the size of set similarity join. The proposed technique relies on a succinct representation of sets using Min-Hash signatures. We exploit frequent patterns in the signatures for the Set Similarity Join (SSJoin) size estimation by counting their support. However, there are overlaps among the counts of signature patterns and we need to use the set Inclusion-Exclusion (IE) principle. We develop a novel lattice-based counting method for efficiently evaluating the IE principle. The proposed counting technique is linear in the lattice size. To make the mining process very light-weight, we exploit a recently discovered Power-law relationship of pattern count and frequency. Extensive experimental evaluations show the proposed technique is capable of accurate and efficient estimation.

1. INTRODUCTION

Given a similarity measure and a minimum similarity threshold, a *similarity join* is to find all pairs of records whose similarity under the measure is greater than or equal to the minimum threshold. Since a set can generalize many data types, the *Set Similarity Join* (*SSJoin*) is a common abstraction of a similarity join problem. For instance, a large scale customer database may have many redundant entries which may lead to duplicate mails being sent to customers. To find candidates of duplicates, addresses are converted into sets of words or n-grams and then an SSJoin algorithm can be used. SSJoin has a wide range of applications including query refinement for web search [28], near duplicate document detection and elimination [5]. It also plays a crucial role in data cleaning process which detects and removes errors and inconsistencies in data [29, 2]. Accordingly, the SSJoin problem has recently received much attention [29, 2, 7, 3, 17, 16].

It is noted that SSJoin is often used as a part of a larger query [2]. In this scenario, a user may want to retrieve all similar pairs possibly conditioned by other predicates or constraints. Thus, it is not a one-time operation and is performed repeatedly with different predicates by users. To handle these general similarity queries, Chaudhuri et al. identified the *SSJoin* operation as a primitive operator for performing similarity joins [7]. The SSJoin operation can also be important in inconsistent databases. Fuxman et al. [15] proposed a system to answer SQL queries over inconsistent databases where the data cleaning operation is performed on-the-fly.

To successfully incorporate the SSJoin operation in relation database systems, it is imperative that we have a reliable technique for the SSJoin size estimation. The query optimizer needs an accurate estimation of the size of each SSJoin operation to produce an optimized query plan. However, to the best of our knowledge, this problem has not been studied previously in the literature. This motivates us to study the SSJoin size estimation problem.

A variety of (dis)similarity measures have been used in the literature such as Jaccard similarity, cosine similarity and overlap threshold [29, 7, 3]. The Jaccard similarity for two sets r, s, $J_S(r, s)$ is defined as $|r \cap s|/|r \cup s|$. It is one of the most widely accepted measures because it can support many other similarity functions [5, 29, 7]. Thus, we focus on the Jaccard similarity for our similarity measure in this paper.

DEFINITION 1 (SSJ PROBLEM). *Given a collection of sets R and a threshold τ on Jaccard similarity J_S, estimate the number of pairs (r, s), $SSJ(\tau)$ such that $J_S(r, s) \geq \tau$, $r, s \in R$ and $r \neq s$.*

Several facts are worth mentioning in the problem formulation. First it corresponds to a self-join case. Although the join between two collections of sets is more general, many applications of SSJoin are actually self-joins: query refinement [28], duplicate document or entity detection [19, 5], or coalition detection of click fraudsters [26]. Naturally, a majority of the SSJoin techniques proposed are evaluated on self-joins [29, 7, 2, 3]. Second, self-pairs (r, r) are excluded in counting the number of similar pairs so that the answers are not masked by the big count, $|R|$. In addition, we do not distinguish between the two different orderings of a pair (i.e., $(r, s) = (s, r)$). It is, however, trivial to adapt the proposed technique to consider the self-pairs or the ordering of pairs in the answer if necessary.

*This research was supported by Basic Science Research Program through the National Research Foundation funded by the Ministry of Education, Science and Technology (grant number 2009-0078828).

VLDB '09, August 24-28, 2009, Lyon, France

Random sampling is a natural technique for the selectivity (or size) estimation problems since it does not suffer from the attribute value independence assumption and is not restricted to equality and range predicates. Recently, the technique of *Hashed Sampling* was proposed for the result size estimation of the set similarity for *selection* queries [17]. It builds a randomly sampled inverted index structure for the selection queries and the selectivity of a query is estimated by performing selection query processing on the samples. Basically join size can be estimated using the Hashed Sampling by applying an SSJoin algorithm on the constructed inverted index. However, it suffers from quadratic processing time and may not be useful for query optimization purposes. Furthermore, we observe that the estimate is highly dependent on the actual samples used (i.e., large variance). This can be compensated by having much larger sample sizes, which makes it impractical for query optimization.

In this paper, we hypothesize that it is possible to perform SSJoin size estimation in a sample-independent fashion. As an overview, our approach works as follows. We first generate Min-Hash signatures for the samples filtered by other predicates. As a succinct representation of a set, min-hash signatures generally result in a faster operations due to its smaller size [11, 5]. Then we perform frequent pattern mining on the signatures. The distribution information of the signature patterns, captured in Power-laws, is used in estimating the size of union. Finally, the estimated size is adjusted so that it correctly reflects the SSJoin size in the original set space. Specifically, we make the following contributions.

- We reduce the SSJ problem to a pattern mining problem on the signatures, which enables very efficient counting of pairs.
- Naive approaches using the signature patterns fail since a pair could be counted multiple times in several patterns. We propose a novel counting technique called *Lattice Counting* which counts the number of pairs satisfying the threshold minimizing overcounting.
- Recently, it was observed in [10] that pattern frequency-count distribution generally follows a Power-law distribution. We exploit this observation to efficiently extract the necessary information for the *Lattice Counting* formula. It is possible because we only need the count information and do not need the actual patterns. A high minimum support threshold can be used so that it is fast enough to be used in query optimization.
- We observe that there is a systematic overestimation when Min-Hashing is used. We establish a model to simulate the count shift. An efficient method for correcting the overestimation is proposed.

This paper is structured as follows. Section 2 lists related work. Section 3 introduces the Min-Hash signature and signature pattern, and gives an overview of the framework. Section 4 presents the Union Formula which gives the SSJoin size based on the mined pattern distribution. Efficient processing based on the Power-law distribution is proposed in Section 5. A procedure to overcome the systematic overestimation by Min-Hashing is introduced in Section 6. The experimental results are presented in Section 7. Section 8 concludes with future directions.

2. RELATED WORK

Hadjieleftheriou et al. studied the problem of selectivity estimation for set similarity queries [17]; they did not consider the join size estimation problem. They proposed the 'Hashed Sampling' technique which performs sampling from the inverted index using Min-Hashing. We extend this technique to the SSJoin problem and compare it with the proposed technique.

Many algorithms have been proposed on the set similarity join (or selection) problem [29, 2, 7, 3, 16]. In general, SSJoin algorithms rely on a nested loop, where the inner loop seeks all the matching sets for each set in the outer loop. As will be shown later, the SSJoin size is in general very skewed and for high similarity thresholds, the selectivity is very high. In this case, we may need relatively bigger samples and the quadratic complexity may make applying SSJoin algorithms on the samples less scalable.

Frequent pattern mining has received considerable attention for the last decade [6]. The proposed technique uses frequent pattern mining algorithms and to the best of our knowledge our work is the first to apply frequent pattern mining to join size estimation problem. Specifically, we use the trie-based Apriori algorithm [14]; virtually any frequent pattern mining algorithms can be used for our estimation purposes since we only rely on the most basic problem formulation [1]. For instance, we can run the frequent pattern mining algorithms with memory-constraints or top-k constraints. See [6] for a survey on this topic.

Power-laws have been observed in a wide range of manmade and natural worlds [30, 9]. They have been successfully used in the literature. e.g., 'query optimizer' [21], 'self-join size estimation' [27], or 'spatial join selectivity' [13]. In relation to frequent pattern mining, Chuang et al. observed that pattern frequency-count distribution follows a Power-law distribution [9]. We exploit this finding in efficient estimation of the SSJoin size.

Recently, several algorithms have been proposed to estimate the number of frequent patterns [4, 9, 22]. Accurate and efficient estimation of the number of frequent patterns is a key factor in our estimation of SSJoin size. Although the proposed technique is primarily based on the observation by Chuang et al. [9], it can benefit from these new developments as well.

Bellman synopsis is a system that performs data mining on the structure of the database [12]. Diverse synopsis structures including Min-Hash signatures are proposed. A join size estimation method using signatures is proposed, but the semantic is different since the similarity between two fields are measured after collapsing all values in fields and the exact join is considered.

3. SIGNATURE PATTERN

3.1 Min-Hash Signature

We first review the Min-Hash signature which has widely been used in similarity related applications [11, 5, 8, 12, 25].

Consider a set of random permutations $\Pi = \{\pi_1, \ldots, \pi_M\}$ on a universe $\mathcal{U} \equiv \{1, \ldots, U\}$ and a set $r \subset U$. Let $min(\pi_i(r))$ denote $min(\{\pi_i(x)|x \in r\})$. Π is called *min-wise independent* if for any subset $r \subset U$ and any $x \in r$, when π_i is chosen at random in Π, we have $Pr(min(\pi_i(r)) = \pi_i(x)) = \frac{1}{|r|}$. Then with respect to two sets r and s, if Π

is min-wise independent, $Pr(min(\pi_i(r)) = min(\pi_i(s))) = J_S(r,s)(= |r \cap s|/|r \cup s|)$ [5, 11]. To estimate J_S (denoted as $\hat{J}_S$), the signature vector of r is constructed as: $sig_r = [min(\pi_1(r)), \ldots, min(\pi_M(r))]$. We call it the *Min-Hash signature* of r. Using the Min-Hash signatures, J_S is estimated with

$$\hat{J}_S(r,s) = \frac{|\{i|mh_i(r) = mh_i(s), 1 \leq i \leq M\}|}{M}. \quad (1)$$

The Min-Hash signature of r is denoted by $sig(r)$ and we will call it the *signature* of r in the remainder of the paper. As truly random permutation π is expensive, hash functions are usually used.

3.2 A Motivating Example

Figure 1 shows an example DB and corresponding signatures, $sig(DB)$. The signature size $M = 4$ and the universe of hash functions is $[1, 5]$. Hash functions are not shown.

DB	
r_1	$\{7, 10, 19, 52, 67\}$
r_2	$\{10, 19, 43, 52\}$
r_3	$\{10, 13, 43, 52, 67, 85\}$
r_4	$\{10, 38, 43, 49, 80, 94\}$
r_5	$\{3, 25, 29, 47, 50, 66, 73, 75\}$

$sig(DB)$	
$sig(r_1)$	$[4, 3, 5, 2]$
$sig(r_2)$	$[4, 3, 3, 5]$
$sig(r_3)$	$[4, 3, 2, 2]$
$sig(r_4)$	$[3, 3, 3, 2]$
$sig(r_5)$	$[1, 1, 1, 3]$

Figure 1: An example DB

Suppose $\tau = 0.5$ is given. Using Equation (1), we can estimate $J_S(r_i, r_j)$ for $r_i, r_j \in DB$ with their signatures. For instance, $\hat{J}_S(r_1, r_2)$= 0.5 since $sig(r_1)$ and $sig(r_2)$ match at two positions (i.e., the first and the second) out of 4 positions. The true similarity, $J_S(r_1, r_2)$ is $\frac{|\{10,19,52\}|}{|\{7,10,19,43,52,67\}|} = 0.5$. Considering all 5 sets in DB, there are 6 pairs $\{(r_1, r_2)$, (r_1, r_3), (r_2, r_3), (r_1, r_4), (r_3, r_4), $(r_2, r_4)\}$ that have at least two matching positions out of 4 positions in their signatures and thus $\hat{J}_S \geq 0.5$.

Using the Min-Hash signatures, SSJoin size estimation can be done by estimating the number of pairs (r, s) such that $sig(r)$ and $sig(s)$ overlap at least at $\lceil \tau \cdot M \rceil$ positions. A naive way to estimate the join size using the signatures is to first build signatures for all sets and then to compute every pairwise similarity. Although similarity computation between two signatures are generally assumed to be cheaper, this approach still has quadratic complexity.

3.3 The FSP Problem

An interesting observation is that *any pair (r, s) from a group of records which shares the same values at least at $\lceil \tau M \rceil$ positions in their signatures satisfy $J_S(r, s) \geq \tau$ with high probability.* For instance, in the above example, r_1, r_2 and r_3 have 4 and 3 at the first and the second signature positions. Any pair from r_1, r_2, r_3 matches at least at those two positions in their signatures. There are 3 such pairs: $(r_1, r_2), (r_1, r_3), (r_2, r_3)$ and they are estimated to have $J_S \geq 0.5$. We formalize this intuition with *signature patterns*.

Definition 2 (Signature Pattern). *A signature pattern is a signature where any of its values is possibly substituted with X which denotes a 'don't care' position. A signature (or a set in the database) matches a signature pattern if its signature values are the same as the pattern values at all positions that are not X. Signature values at positions marked by X do not matter. For a pattern, sig, we define the following notations.*

signature pattern	length	matching sets	freq.
$[4, 3, X, X]$	2	r_1, r_2, r_3	3
$[4, X, X, 2]$	2	r_1, r_3	2
$[X, 3, 3, X]$	2	r_2, r_4	2
$[X, 3, X, 2]$	2	r_1, r_3, r_4	3
$[4, 3, X, 2]$	3	r_1, r_3	2

Figure 2: An example of signature patterns ($freq \geq 2$)

- *length, $len(sig)$: the number of non-X values in the pattern;*
- *frequency, $freq(sig)$: the number of sets in the database whose signatures match the pattern.*

Example 1. *Figure 2 shows signature patterns with $freq \geq 2$ found in the example of Figure 1 with their lengths, matching sets and frequencies. For instance, $len([4, 3, X, X]) = 2$ since it has two non-X values. Its frequency is 3 because three sets, r_1,r_2,r_3 match the pattern.*

Based on the signature pattern, we define the Frequent Signature Pattern problem as follows.

Definition 3 (FSP Problem). *Given a threshold t on the pattern length, $FSP(t)$ is the number of pairs which share a pattern sig such that $len(sig) \geq t$ and $freq(sig) \geq 2$.*

To solve the $SSJ(\tau)$ problem, we solve the $FSP(t)$ problem where $t = \lceil \tau M \rceil$. In the sequel, we implicitly assume a fixed value of M. If we view the signature of each set as a transaction and *each signature value along with its position* as an item, we can apply traditional frequent pattern mining algorithms to the FSP problem. Given τ we discover the signature patterns with $length \geq \lceil \tau M \rceil$ and $freq \geq 2$. A minimum frequency of 2 is necessary to produce a pair. If the frequency of a pattern is k, there are at least $npair(k) \equiv \binom{k}{2} = k(k-1)/2$ pairs satisfying the similarity threshold τ in the database. Exploiting this observation, we can have the following two step approach in estimating the SSJoin size.

1. Compute the frequent signature patterns with $length \geq \lceil \tau M \rceil$ and $freq \geq 2$.
2. For each pattern of frequency k, count the number of pairs by $npair(k) = k(k-1)/2$, and then aggregate all the number of pairs.

However, this approach has several critical challenges.

- There are overlaps among the signature patterns. Consider $sig_1 = [4, 3, X, X]$, $sig_2 = [X, 3, X, 2]$. If $\tau = 0.5$, both patterns should be considered in the answer since their $length \geq 2 = 0.5 \cdot 4$. There are 3 pairs from sig_1 and 3 pairs from sig_2 satisfying $\tau = 0.5$: (r_1, r_2), (r_1, r_3), (r_2, r_3) from sig_1 and (r_1, r_3), (r_1, r_4), (r_3, r_4) from sig_2. If we simply add the two counts 3 and 3, we are counting the pair (r_1, r_3) twice. Without consideration of this overlap, the estimate could be a large overestimation.
- We need all signature patterns with $freq \geq 2$. It is equivalent to setting the minimum support threshold to 2 in the frequent pattern mining problem. However, the search space for such a low threshold is not generally feasible.

We will address the first issue in the next section and the second issue in Section 5.

4. LATTICE COUNTING

4.1 Computing The Union Cardinality

Let $S_p(sig)$ denote the set of pairs not including the self-pairs that match with a pattern sig. We call $S_p(sig)$ the *matching set* of sig. Let I_ℓ denote the index set of all the signature patterns in DB such that $length \geq \ell$ and $freq \geq 2$. The SSJoin size, $SSJ(\tau)$, is the same as $FSP(t)$ which is the number of pairs that match any pattern $sig_i, i \in I_t$, where $t = \lceil \tau \cdot M \rceil$. As a pair (r, s) can match with multiple signature patterns, we need to compute the cardinality of the *union* of the set of pairs as follows. Given τ,

$$FSP(t) = |\bigcup_{i \in I_t} S_p(sig_i)| \quad where \quad t = \lceil \tau \cdot M \rceil . \qquad (2)$$

EXAMPLE 2. *Consider Example 1 with $\tau = 0.5$ and $M = 4$. Every signature pattern with $freq \geq 2$ and length $\geq 0.5 \cdot 4$ contributes to $FSP(2)$. As shown in Figure 2, there are 5 such patterns in DB, we have $I_2 = \{1, 2, 3, 4, 5\}$ and can compute the size from Equation (2) as follows.*

sig_i	$S_p(sig_i)$ or S_i (*matching set*)
$sig_1 = [4, 3, X, X]$	$\{(r_1, r_2), (r_1, r_3), (r_2, r_3)\}$
$sig_2 = [4, X, X, 2]$	$\{(r_1, r_3)\}$
$sig_3 = [X, 3, 3, X]$	$\{(r_2, r_4)\}$
$sig_4 = [X, 3, X, 2]$	$\{(r_1, r_3), (r_1, r_4), (r_3, r_4)\}$
$sig_5 = [4, 3, X, 2]$	$\{(r_1, r_3)\}$

$$SSJ(0.5) = FSP(2) = |S_p(sig_1) \cup \cdots \cup S_p(sig_5)| = 6$$

If we ignore the overlap, $FSP(t)$ is $\sum_{i \in I_t} |S_p(sig_i)|$. We call this naive strategy the *Independent Sum* (IS) method. The correct way of computing cardinality of the union is to use the set Inclusion-Exclusion (IE) principle. However, the IE formula has exponential complexity in the number of sets.

4.2 The Union Formula Exploiting Lattice

Consider an example of $FSP(2)$ which computes $|S_p(sig_1) \cup S_p(sig_2) \cup S_p(sig_4)|$. Let us use S_i to denote $S_p(sig_i)$ for simplicity. Using the IE principle, $FSP(2)$ is computed in the following way:

$$|S_1| + |S_2| + |S_4| - (|S_1 \cap S_2| + |S_1 \cap S_4| + |S_2 \cap S_4|) + |S_1 \cap S_2 \cap S_4| = 3 + 1 + 3 - (1 + 1 + 1) + 1 = 5.$$

Interestingly, if we look into all the intersections, we see that many of them result in the same set: $S_1 \cap S_2 = S_1 \cap S_4 = S_2 \cap S_4 = \{(r_1, r_3)\}$.

We can structure this overlap with the semilattice shown in Figure 3. The patterns and their matching sets are organized using lattices (or semilattices). The edges represent the inclusion relationship. The *level* of a pattern is the number of non-X values and the level of a matching set is the level of the corresponding pattern. For instance, S_5 has edges to S_1 and S_2 and $S_1 \cap S_2 = S_5$. S_5 has three children and an intersection between any two children will result in S_5. This implies that S_5 appears three times in the intersections between two sets in the IE formula since there are three ways of choosing two children out of three. Likewise, it appears once in the intersections among three sets, $S_1 \cap S_2 \cap S_4$. The net effect is that S_5's contribution to the IE formula is $|S_5|$ multiplied by $(-3+1)$. The negative sign of -3 is from the alternating signs in the IE formula. Incorporating this observation, the above IE formula is simplified

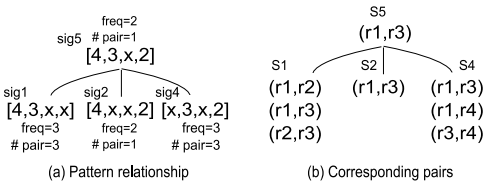


Figure 3: Overlapping relationship

as follows without actually performing any intersections.

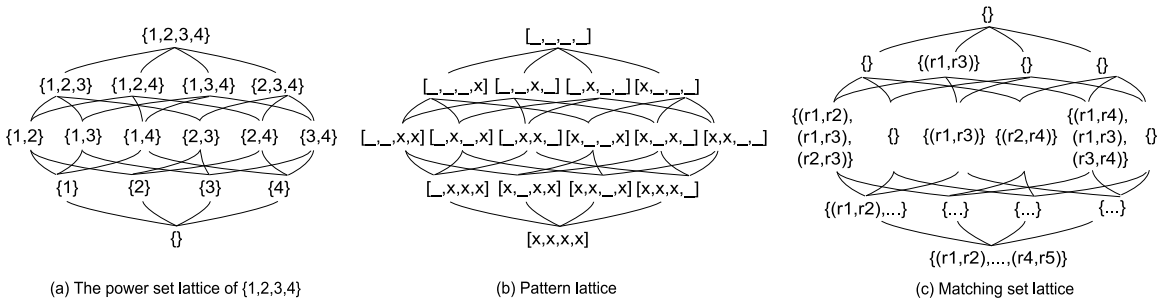
$$|S_1| + |S_2| + |S_2| + (-3 + 1)|S_5| = 3 + 1 + 3 - 2 \cdot 1 = 5.$$

Exploiting the lattice structure, each count ($|S_p(sig_i)|$) is processed exactly once whereas the IE method computes every intersection unaware of the underlying structure. Notice that this approach only involves individual quantities, $|S_p(sig)|$, which can be acquired by pattern mining through $|S_p(sig)| = npair(freq(sig)) = \binom{freq(sig)}{2}$. From this example, we make the following crucial observations.

- Signature patterns and their matching sets can be organized with the lattice structure. A node in the pattern lattice represents a signature pattern and a node in the matching set lattice represents the set of matching pairs for each pattern.
- The $FSP(t)$ problem can be solved by finding the cardinality of the union of matching sets at level t. In Figure 3, $FSP(2)$ is the cardinality of the union of matching sets at level 2, $|S_1 \cup S_2 \cup S_4|$.
- The multiplicity of a set in the IE formula is expressed by a *coefficient*. The coefficient only depends on the lattice structure. For instance, in the above example, $|S_5|$ has a coefficient of -2.

Let us generalize this intuition. The pattern lattice and the matching set lattice are isomorphic to a power set lattice, $2^{\{1,\ldots,M\}}$. Figure 4 shows the lattice structures for the database in Figure 1. Figure 4 (a) depicts the underlying power set lattice, $2^{\{1,\ldots,4\}}$. Figure 4 (b) shows the pattern lattice. The pattern lattice needs some modifications; a pattern in the pattern lattice is a generalization of a signature pattern where '_' denotes *matching* positions and 'X' denotes *don't care* positions. Figure 4 (c) is the matching set lattice corresponding to the pattern lattice for the database in Figure 1. A node in the matching set lattice represents the set of pairs whose signatures match at the positions marked '_' in the corresponding pattern. The specific matching values are not important. For instance, $[\_, \_, X, X]$ is a pattern selecting all pairs which match at their first and second positions, and $\{(r_1, r_2), (r_1, r_3), (r_2, r_3)\}$ are the corresponding matching sets. If there were r_6, r_7 such that $sig(r_6) = [1, 1, 6, 2]$ and $sig(r_7) = [1, 1, 2, 4]$ then (r_6, r_7) would have been in the matching set as well.

Consider the coefficient of node $[\_, \_, \_, X]$ for $FSP(2)$. The sublattice structure whose top is $[\_, \_, \_, X]$ is the same as the lattice in Figure 3 (a), and thus we can infer it has the same coefficient. In other words, it appears the same number of times in the IE formula for union cardinality of sets at level 2. Moreover, observe that all the other nodes at level 3 have the identical structure, which gives the same coefficient. We can see that all nodes at the same level will have



(a) The power set lattice of {1,2,3,4} (b) Pattern lattice (c) Matching set lattice

Figure 4: An example pattern lattice structure

the same coefficient since there will be the same number of ways to choose a certain number of nodes at a level. Thus, we can work with the sum of the cardinalities of matching sets at each level since they all have the same coefficient. Let the sum at level ℓ be F_ℓ. We call it the *level sum*. Define the index set $\mathcal{I}_\ell$ that lists all patterns at level ℓ, then

$$F_\ell = \sum_{i \in \mathcal{I}_\ell} |S_p(p_i)|. \tag{3}$$

Using the level sum, the computation is as follows.

$$\begin{aligned} F_2 &= |S_p(sig_1)| + |S_p(sig_2)| + |S_p(sig_3)| + |S_p(sig_4)| \\ F_3 &= |S_p(sig_5)| \\ FSP(2) &= C_2F_2 + C_3F_3 = 1 \cdot (3+1+1+3) + (-2) \cdot 1 = 6 \end{aligned}$$

Formalizing this intuition, we define the pattern lattice structure and the Lattice Counting (LC) problem as follows.

Definition 4 (Pattern Lattice Structure). *Given a set collection of sets R and the signature size M, the pattern lattice structure is a tuple $\mathcal{L} = (\mathcal{M}, \mathcal{L}_\mathcal{P}, \mathcal{L}_R)$ where $\mathcal{M} = \{1, ..., M\}$, $\mathcal{L}_\mathcal{P}$ is the pattern lattice and $\mathcal{L}_R$ is the matching set lattice. For each $m \subset \mathcal{M}$, the corresponding $p \in \mathcal{L}_\mathcal{P}$ is a signature pattern such that p has '_' in the position $i \in m$ and 'X' in the other positions. For each $p \in \mathcal{L}_\mathcal{P}$, $S_p(p) \in \mathcal{L}_R$ is the set of pairs $(r, s), r, s \in R$ such that $sig(r)$ and $sig(s)$ have the same values at position marked '_' in p.*

The partial order $\leq$, the least upper bound $\vee$, and the greatest lower bound $\wedge$ of $x, y \in \mathcal{L}_\mathcal{P}$ are defined by $\subseteq$, $\cup$ and $\cap$ between the corresponding subsets in the power set respectively. For any $x, y, z \in \mathcal{L}_\mathcal{P}$ the following conditions hold.

- *Inclusion*: $x \leq y$ *iff* $S_p(x) \supseteq S_p(y)$
- *Refinement*: $x \vee y = z$ *iff* $S_p(x) \cap S_p(y) = S_p(z)$

Definition 5 (LC Problem). *Given $\mathcal{L} = (\mathcal{M}, \mathcal{L}_\mathcal{P}, \mathcal{L}_R)$ and a level threshold t, the LC problem is to estimate the cardinality of union of matching sets at level t in $\mathcal{L}_R$.*

$$LC(t) = |\bigcup_{i \in \mathcal{I}_t} S_p(p_i)|$$

We can think of the $LC(t)$ problem as the $FSP(t)$ problem defined on the lattice. We simplify the IE formula as the sum of the cardinality of each matching set $S \in \mathcal{L}_{\mathcal{DB}}$ multiplied by some coefficient. As all the matching sets at the same level have the same coefficient, we can compute $LC(t)$ using $\sum_{l=t}^{M} C_{\ell,t} \cdot F_\ell$. We give the exact answer for $LC(t)$ for an arbitrary pattern lattice structure in the next equation.

$$\begin{aligned} LC(t) &= \sum_{l=t}^{M} C_{\ell,t} \cdot F_\ell, where \\ C_{\ell,t} &= \sum_{r=2}^{\binom{M}{t}} (-1)^{r+1} B_{\ell,t,r} \\ B_{\ell,t,r} &= \sum_{i=0}^{i<\ell-t} (-1)^i \cdot \binom{\ell}{i} \cdot \binom{\binom{\ell-i}{\ell-t-i}}{r} \end{aligned} \tag{4}$$

The coefficient C can be computed by counting how many times a node appears at all intersections of size $r \geq 2$ in the IE formula. B keeps track of the occurrence at each r. The intermediate derivation depends on the inclusion and the refinement properties.

A similar lattice structure is used to estimate the selectivity of string matching with edit distance [24]. Because the focus of [24] is on low edit distance threshold, the lattice used there is shallow and small in size. In contrast, the lattice structure defined here is entirely new and represents the whole set database. It leads to computational challenges not seen in [24].

4.3 Level Sum Computation

Note that in Equation (4), only F_ℓ depends on the data. So computing $LC(t)$ reduces to computing the level sums $F_\ell, t \leq \ell \leq M$. F_ℓ is counted as follows.

$$F_\ell = \sum_{i \in \mathcal{I}_\ell} |S_p(p_i)| = \sum_{i \in \mathcal{I}_\ell} npair(freq(p_i)) \tag{5}$$

The right hand side of Equation (5) uses the frequencies of all the signature patterns of length ℓ, and $\mathcal{I}_\ell$ lists all such patterns. We can simplify the F_ℓ computation by noting that patterns with the same $freq$ value will have the same $npair(freq(p_i))$ value. Rather than repeatedly evaluating the *npair* function for the same frequency, a better strategy will be to count how many patterns have the frequency of f for each f and to evaluate *npair* just once for each f. As there are a large number of patterns at low frequencies this reduction could be huge. Let us group all patterns $p_i, i \in \mathcal{I}_\ell$ by their frequencies. Let $\mathcal{I}_{l,f}$ denote the index set for the patterns such that $|p_i| = l$ and $freq(p_i) = f$ if and only if $i \in \mathcal{I}_{l,f}$. For instance, $\mathcal{I}_{2,3}$ lists all the length-2 patterns with frequency of 3. In the example $\mathcal{I}_{2,3} = \{1, 4\}$ since $|sig_1| = |sig_4| = 2$ and $freq(sig_1) = freq(sig_4) = 3$ (see Figure 2). $\mathcal{I}_{l,f}$ defines a partitioning over $\mathcal{I}_\ell$ and $|\mathcal{I}_{l,f}|$ is the number of patterns with $freq = f$ in the level ℓ. If we use

$mf(\mathcal{I}_\ell)$ for $max(freq(x_i)), i \in \mathcal{I}_\ell$,

$$F_\ell = \sum_{2 \le f \le mf(\mathcal{I}_\ell)} npair(f) \cdot |\mathcal{I}_{l,f}| \ . \quad (6)$$

A frequent pattern mining algorithm gives all the frequent patterns above the minimum support threshold. We extract the count information grouped first by pattern length and then by pattern frequency. This gives us $|\mathcal{I}_{l,f}|$ for all l and f found in the database. For a specific ℓ, we call the set of tuples $(f, |\mathcal{I}_{l,f}|)$ *pattern distribution* for level ℓ.

EXAMPLE 3. *The next table is the signature pattern distribution for the running example.*

pattern distribution				
level ℓ	f (freq)	$\lvert\mathcal{I}_{\ell,f}\rvert$	sig. pattern	matching sets
2	2	2	$[4, X, X, 2]$	r_1, r_3
			$[X, 3, 3, X]$	r_2, r_4
	3	2	$[4, 3, X, X]$	r_1, r_2, r_3
			$[X, 3, X, 2]$	r_1, r_3, r_4
3	2	1	$[4, 3, X, 2]$	r_1, r_3

$|\mathcal{I}_{2,2}| = 2$ *since two signature patterns,* $[4, X, X, 2]$ *and* $[X, 3, 3, X]$*, have two matching sets* ($f = 2$)*. Likewise* $|\mathcal{I}_{2,3}| = 2$ *since two signature patterns have three matching sets* ($f = 3$)*. For* F_2*, each of the two patterns with* $f = 2$ *generates one pair* ($\binom{2}{2} \cdot 2 = 2$)*, and each of the two patterns with* $f = 3$ *generates three patterns* ($\binom{3}{2} \cdot 2 = 6$)*. So* $F_2 = 2 + 6 = 8$*. The whole level sum computation is as follows.*

$$F_2 = \sum_{2 \le f \le 3} npair(f) \cdot |\mathcal{I}_{2,f}| = \binom{2}{2} \cdot 2 + \binom{3}{2} \cdot 2 = 8$$

$$F_3 = \sum_{2 \le f \le 2} npair(f) \cdot |\mathcal{I}_{3,f}| = \binom{2}{2} \cdot 1 = 1$$

After computing the level sums, the rest of the LC computation is summing those level sums multiplied by coefficients in Equation (4). $SSJ(0.5) = LC(2) = 6$ *is computed as follows.*

level ℓ	F_ℓ	$C_{4,\ell}$	$C_{4,\ell} \cdot F_\ell$
2	$F_2 = 8$	1	$1 \times 8 = 8$
3	$F_3 = 1$	-2	$-2 \times 1 = -2$
4	$F_4 = 0$	3	$0 \times 3 = 0$
$LC(2) = \sum_{\ell=2}^{4} C_{4,\ell,2} \cdot F_\ell$			6

5. POWER-LAW BASED ESTIMATION

Note from Equation (6) that we only need the frequency of patterns *not the actual patterns* for our estimation purposes. Thus, our framework only *counts* patterns and does not exactly generate and store patterns. Equation (6) also requires the distribution of all signature patterns with $freq \ge 2$. Most frequent pattern mining algorithms are not designed to handle such a low support threshold. Even if they could, it would take too long to be used for the query optimization task. In this section, we address this issue and show how we can efficiently compute Equation (6).

5.1 Level Sum with Power-Law Distribution

Chung et al. recently discovered that a Power-law relationship is found in the *pattern support distribution* [9]. A Power-law is a special relationship between two quantities. It is found in many fields of natural and manmade worlds. A quantity x obeys a Power-law if it is drawn from a probability distribution $p(x) \propto x^\alpha, \alpha < 0$ [10]. The Zipf distribution is one of related discrete Power-law probability distributions and it characterizes the "frequency-count" relationship [9]. When c_i is the count of distinct entities that appear f_i times in the data set, the distribution is described by

$$c_i = f_i^\alpha \cdot 10^\beta. \quad (7)$$

Chung et al's hypothesis refers to the distribution of the pattern count versus the pattern frequency.

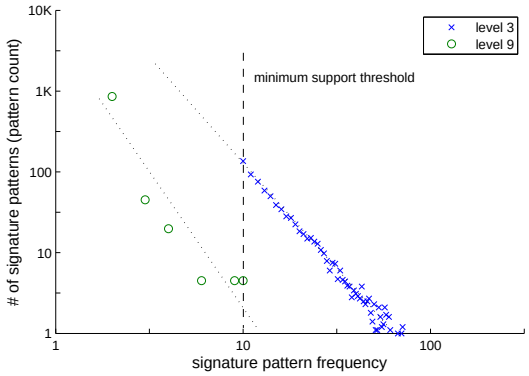


Figure 5: Signature pattern distribution

Figure 5 plots the signature pattern distributions of level 3 and 9 in the DBLP data set when $M = 10$. The x-axis is pattern frequency and y-axis is pattern count. For instance, x=10, y=121 for level 3 in the plot means there are 121 length-3 patterns that appear 10 times in the DB. The pattern count (y-value) decreases as the frequency (x-value) increases; a majority of patterns appear only in a small number of sets and a few patterns appear in many sets. We can also observe that the points of a higher level (e.g., level 9) distribution are on the lower side of points of a low level (e.g., level 3) distribution. This is because when a pattern is frequent, its sub-patterns are at least as frequent.

This distribution corresponds to $(f, |\mathcal{I}_{l,f}|)$ for each level ℓ. If we have this information for all $f \ge 2$ and ℓ, we have the exact level sums by Equation (6) and thus the exact answer for the SSJ problem. Consider a practical minimum support threshold of $\xi > 2$, e.g., $\xi = 10$ in Figure 5. It is shown as the vertical line in Figure 5. The problem of having $\xi > 2$ is that we will miss the pattern distribution on the left side of the vertical line, i.e., patterns with $2 \le freq < \xi$, because the frequent pattern mining algorithm will only find patterns with $freq \ge \xi$. For patterns at level 3, we see the pattern distribution on the left of the ξ line is missing.

We address the problem of the missing pattern distribution with the Power-law distribution. In accordance with Chung et al's hypothesis, a linear relationship between the pattern count and the frequency is observed in Figure 5. Exploiting this relationship, the strategy is as follows.

1. Find frequent patterns with $\xi > 2$ for efficiency.
2. Estimate the parameters for the Power-law distribution at each level with the acquired patterns.
3. Compute the necessary level sums with the pattern distribution based on the estimated parameters.

Algorithm 1 Power-Law Estimation

Input: A collection of set DB, a Jaccard threshold τ, minimum support ξ, Min-Hash functions $\{mh_1, ..., mh_M\}$

1: build $sig(DB)$ by computing $sig(s)$ for all $s \in DB$
2: run a frequent pattern mining algorithm on $sig(DB)$ with ξ
3: **for all** $1 \leq \ell \leq M$ **do**
4: estimate α_ℓ and β_ℓ (using a linear regression)
5: **end for**
6: $t = \lceil \tau \cdot M \rceil$
7: Est = 0
8: **for all** $t \leq \ell \leq M$ **do**
9: $F_\ell = \sum_{2 \leq f \leq mf(\mathcal{I}_\ell)} npair(f) \cdot f^{\alpha_\ell} \cdot 10^{\beta_\ell}$
10: Est += $C_{\ell,t} \cdot F_\ell$
11: **end for**
12: **return** Est

Our algorithm is independent of the specific choice of the frequent pattern mining algorithm. This gives an opportunity to exploit the extensive studies in the literature.

Algorithm 1 outlines this approach. At line 4, we use linear regression based on the least square fitting to estimate the parameters of the Power-law distribution α_ℓ and β_ℓ for each level ℓ. Other methods such as maximum likelihood estimation could be more reliable, but we use linear regression as in [9] since it is very efficient and is sufficient for our purposes. See [9, 10] for more discussions on this topic.

The next challenge is that we may not have enough points for estimating the parameters when we use a high ξ. This is likely for the distribution at higher levels since patterns at higher levels (longer patterns) are fewer than the ones at lower levels. For instance, we cannot estimate the distribution at level 9 in Figure 5 since most of the points are on the left side of the ξ line and we cannot estimate the parameters. Equation (4) requires all the level sum $F_\ell, t \leq \ell \leq M$. For example, when $M = 10$, $LC(5)$ needs F_5, $F_6, ..., F_{10}$. We present our solution below.

5.2 Approximate Lattice Counting

To compute $LC(t)$, we use the pattern lattice structure to consider all the intersections among the patterns at level t. In the lattice, nodes at a higher level represent more complex relationships among nodes at level t. Observe that the counts of those complex intersections are relatively small as we can infer from Figure 5; the pattern counts of higher levels are much smaller than those of lower levels. Based on this intuition, we employ a *truncation heuristic* which considers only parts of the lattice ignoring some of the complex intersections in the IE formula. Consider $\{1, 2, 3, 4\}$

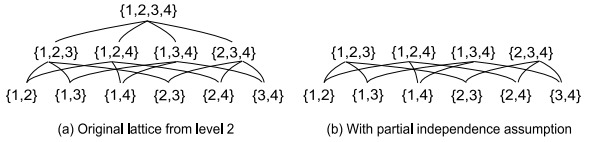
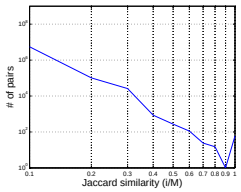
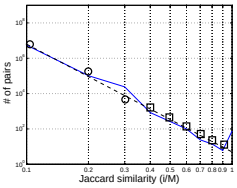


Figure 6: Relaxed lattice

in Figure 6 (a). It represents complex intersections such as $\{1, 2\} \cap \{1, 3\} \cap \{3, 4\}$. If we ignore such higher level intersections, we have a part of the original lattice as in Figure 6 (b) where we only consider intersections between two nodes. Intersections like $\{1, 2\} \cap \{1, 3\} \cap \{3, 4\}$ are ignored in the relaxed lattice structure. Generalizing this intuition, we define *Max-k relaxed lattice* where given $LC(t)$, we only consider nodes between level t and $min(M, t + k)$ ignoring more complicated intersections at level $t + k + 1$ or higher. This gives us the relaxed form of Equation (4).



(a) First subset



(b) Estimation of $T(i)$ in Second subset

Figure 7: # pair-similarity plot of 2 subsets of the DBLP data set

$$LC_k(t) = \sum_{\ell=t}^{K} \hat{C}_{\ell,t}(k) \cdot F_\ell, \quad where \tag{8}$$

$$\hat{C}_{\ell,t}(k) = \sum_{r=2}^{\binom{K}{t}} (-1)^{r+1} B_{\ell,t,r}$$

$$K = min(M, t + k)$$

Notice the replacement of M by K in the summations. We now only sum up to the level $t+k$. This approximate Union Counting formula lets us estimate the SSJoin size when not all the signature lines are available. With $LC_k(t)$ where k is the relaxing parameter, we only need F_ℓ for $t \leq \ell \leq t + k$. For instance, when $M = 10$, $LC_2(5)$ can be computed with only F_5, F_6 and F_7 without F_8, F_9 and F_{10}.

5.3 Estimation With Limited Pattern Distribution

Depending on the available pattern distributions and the similarity threshold, the approximation formula may not be enough to answer $SSJ(t)$. Assume that signature size $M = 10$ and we acquire pattern distributions up to level 6: $F_0, ..., F_6$. Using the Max-2 relaxed lattice, we can compute $LC_2(4)$ since it only needs F_4, F_5 and F_6. However, we still cannot answer $LC(t), t \geq 7$ since we do not have $F_\ell, \ell \geq 7$. Recall that the value of t corresponds to the Jaccard similarity threshold. For instance, when $M = 10$, $t = 7$ corresponds to $\tau = 0.7$. Thus, we still cannot handle a threshold such as 0.7 or higher. For this situation, we make use of a new observation found in a skewed distribution.

Let $T(i)$ denote the number of pairs (r, s) such that $J_S(r, s) = i/M$ with rounding. The difference with $LC(i)$ is that $LC(i)$ counts pairs (r, s) such that $J_S(r, s) \geq i/M$. Thus $LC(t) = \sum_{\ell=t} T(\ell)$. Figure 7 (a) and (b) plot the number of pairs versus the Jaccard similarity in log-log scale in two randomly selected $80K$ subsets of the DBLP data set. That is it plots $(i, T(i))$. Again a Power-law relationship is observed. This result may be of independent interest but it is beyond the scope of this paper.

We can estimate the missing $T(i)$ values using a linear regression. Figure 7 (b) shows an example. If we acquired pattern distributions up to level 6, we can compute $LC(0), ..., LC(4)$ with the Max-2 relaxed lattice. Since $T(i) = LC(i) - LC(i + 1)$, we have $T(0), T(1), T(2), T(3)$. These are depicted as circles in Figure 7 (b). We use these points to estimate the missing $T(i)$'s which are shown as rectangles: $T(4), ..., T(10)$. Now we can answer, for example, $LC(7) = T(7) + ... + T(10)$.

This enables us to estimate SSJoin size even when we have very limited pattern distribution and the Equation (8) cannot be applied. Parameter estimation will produce poor estimations if the initial estimations produced by the Lattice Counting, which are the circles in Figure 7 (b), are not accurate. Section 7 verifies the hypothesis and the accuracy of Lattice Counting.

6. CORRECTION OF THE ESTIMATION

6.1 Systematic Overestimation By Min-Hash

While Min-Hash signatures present several benefits, they can cause overestimation. Figure 8 shows the SSJoin size by the original sets and by their Min-Hash signatures in the DBLP data set. We see that the overestimation by Min-Hash is rather huge. Figure 9 log-plots the true SSJoin size and the size by Min-Hashing. A clear distribution shift is observed. This is problematic since similarity thresholds between 0.5 and 0.9 are typically used [3]. One may suspect that the hash collision is the problem. However, through experiments, we found that this is not the dominating cause as long as the hash domain is not too small. We study the cause of this systematic overestimation and propose a correction procedure below.

Figure 8: SSJoin Size Distribution in the DBLP data

J_S	True # pairs	# pairs by Min-Hash	Relative Error
0	3,167,244,255	3,167,244,255	0.00%
0.1	22,750,745	306,062,044	1245.28%
0.2	577,313	51,556,984	8830.51%
0.3	128,078	6,470,509	4952.01%
0.4	5,634	587,761	10332.39%
0.5	2,049	55,623	2614.64%
0.6	980	6,597	573.16%
0.7	495	1,477	198.38%
0.8	384	645	67.97%
0.9	298	419	40.60%
1	286	318	11.19%

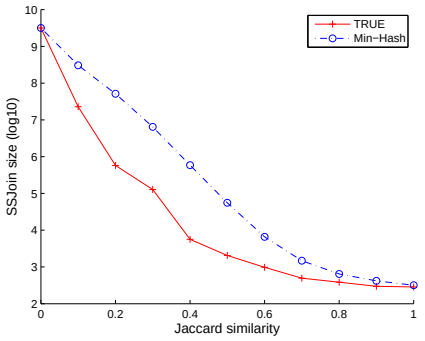


Figure 9: SSJoin size: True vs. MinHash

6.2 Error Correction By State Transition Model

Recall that $T(i)$ denotes the *true* number of pairs (r,s) such that $J_S(r,s) = i/M$. When $J_S(r,s) = i/M$, we call the pair is *in state* i assuming the signature size M is fixed. We define $O(i)$ to be the number of *observed* pairs (r,s) such that $J_S(r,s) = i/M$. Let us define two vectors T and O as $T = [T(0),...,T(M)]^T$ and $O = [O(0),...,O(M)]^T$. The *state transition* of a pair from state i to state j means that the pair's true similarity is i/M but is estimated to be j/M. For example, if $M = 4$ and there are 100 pairs such that $J_S = 2/4$, we say that 100 pairs are in state 2. If there is a 30% chance of state transition from state 2 to 3, 30 of them are falsely estimated to have $J_S = 3/4$. The key point is that when the distribution of T is skewed, it can cause overestimation in O.

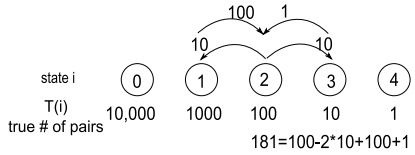


Figure 10: An example of imbalance in transitions

EXAMPLE 4. *Assume $M = 4$ and the distribution of $T(i)$ is as in Figure 10. For instance, 100 pairs have $J_S = 2/4$ and they are in state 2. There exist state transitions since Min-Hash is probabilistic. In the example of Figure 1, $J_S(r_3, r_4) = 1/5$ so the pair's true state is 1 (rounding up 1/5 to 1/4). However with the signatures, it is estimated as $\hat{J}_S(r_3, r_4) = 2/4$ and is in state 2. Suppose there is a 10% of state transition probability to each neighboring state. If the distribution of $T(i)$ is skewed, it causes unbalance in the transitions. For instance, although $T(2) = 100$, $O(2) = 181$ since 20 pairs go to state 1 or 3, and 100 pairs come from state 1, and 1 pair comes from state 3.*

Given a pair (r,s) such that $J_S(r,s) = i/M$, let I denote its state i, and J denote the actual number of matching positions in $sig(r)$ and $sig(s)$. At each position, the matching probability $Pr(\pi_k(r) = \pi_k(s))$ is i/M and all π_k are independent. Thus J is a binomial random variable with parameter i/M. That is

$$Pr(J = j|I = i) = \binom{M}{j} (\frac{i}{M})^j (1 - \frac{i}{M})^{M-j}. \quad (9)$$

If we use $P(j|i)$ for $Pr(J = j|I = i)$, $Pr(j|i)$ is the probability of having j matching positions when the true number of matching positions is i. The observed count $O(j)$ is the sum of all $T(i)$ multiplied by $P(j|i)$ which is moving in mass from state i. Thus $\sum_{i=0}^{M} Pr(j|i)T(i) = O(j)$. This gives us a system of equations for $0 \le j \le M$. Define the matrix A as $A_{j,i} = Pr(j|i)$. The system is described by

$$AT = O. \quad (10)$$

EXAMPLE 5. *When $M = 4$, based on Equation (9), A is as follows.*

$$A = \begin{pmatrix} P(0|0) & P(0|1) & P(0|2) & P(0|3) & P(0|4) \\ P(1|0) & P(1|1) & P(1|2) & P(1|3) & P(1|4) \\ P(2|0) & P(2|1) & P(2|2) & P(2|3) & P(2|4) \\ P(3|0) & P(3|1) & P(3|2) & P(3|3) & P(3|4) \\ P(4|0) & P(4|1) & P(4|2) & P(4|3) & P(4|4) \end{pmatrix}$$

Then by Equation (10), $AT = O$ becomes

$$\begin{pmatrix} 1.000 & 0.316 & 0.063 & 0.004 & 0 \\ 0 & 0.422 & 0.250 & 0.047 & 0 \\ 0 & 0.211 & 0.375 & 0.211 & 0 \\ 0 & 0.047 & 0.250 & 0.422 & 0 \\ 0 & 0.004 & 0.063 & 0.316 & 1.000 \end{pmatrix} \begin{pmatrix} T(0) \\ T(1) \\ T(2) \\ T(3) \\ T(4) \end{pmatrix} = \begin{pmatrix} O(0) \\ O(1) \\ O(2) \\ O(3) \\ O(4) \end{pmatrix}.$$

Let us analyze $O(3)$. $0 \cdot T(0) + 0.047 \cdot T(1) + 0.25 \cdot T(2) + 0.422 \cdot T(3) + 0 \cdot T(4) = O(3)$. 4.7% of $T(1)$ constributes to $O(3)$. 25% of $T(2)$ goes to $O(3)$ and 42.2% of $T(3)$ goes to $O(3)$.

Algorithm 2 LC with Error Correction

Procedure LCWithErrorCorrection
Input: similarity threshold t, max iteration count θ
Output: $LC(t)$
1: Compute $LC(\ell)$ for $1 \leq \ell \leq M$ (estimates without correction)
2: $O(0) = N \cdot (N-1)/2, O(M) = LC(M)$ // N: data set size
3: $O(\ell) = LC(\ell) - LC(\ell+1), 1 \leq \ell < M$
4: SanityCheck($LC(\ell), O(\ell), 1 \leq \ell \leq M$) // desc. in Sec 7.1
5: PowerLawInterpolate(O)
6: minimize $\|WAX - WO\|$ with θ, subject to $X \geq 0$
7: PowerLawInterpolate(X)
8: **return** $LC(t) = \sum_{\ell=t}^{M} X(\ell)$

Procedure PowerLawInterpolate
Input: vector $V = [V_1, ..., V_k]$
Output: modified $V = [V_1, ..., V_k]$
1: Estimate Power-law parameter α and β from (ℓ, V_ℓ)
2: **for all** $V(\ell) \leq 0$ **do**
3: $\quad V(\ell) = \ell^{\alpha} \cdot 10^{\beta}$
4: **end for**

The huge incoming counts from $T(i)$ to $O(j), i < j$ can inflate $O(j)$. Since $\hat{LC}(t) = \sum_{i=t}^{M} O(i)$, without appropriate correction, $LC(t)$ is doomed to be a massive overestimation. A tempting option is to compute T by solving the system of linear equations. A is non-singular and only depends on the choice of M. The join size is computed by $\hat{T} = A^{-1}O$.

However, this simple approach does not work for two reasons. First, $\hat{T}$ might have negative values. Second, O is highly skewed: $O(i) \gg O(j)$ when $i < j$. So lower entries in O, say $O(0), O(1)$ and $O(2)$, will have a dominating effect making higher entries negligible.

Thus, rather than solving the system of linear equations, we solve a NNLS (non-negative least square) constrained optimization problem [23]. To prevent the solution from being dominated by lower entries, we scale the matrix by a weight matrix W so that higher entries in O will have approximately the same effect in the least square solution. We could use W defined as $W_{i,i} = 1/O(i)$ and $W_{i,j} = 0, i \neq j$. In summary, the final step of our estimation corrects the estimation by solving the next NNLS problem.

$$\text{minimize} \quad \|WAX - WO\| \quad \text{subject to} \quad X \geq 0 \qquad (11)$$

The dimension of the matrix of interest is not big, say 10, and we can solve the system fairly efficiently. (i.e., milliseconds)

The NNLS may fail for several reasons. It may diverge or have zero estimates in X. We use the technique in Section 5.3 in these cases. Missing T or X values are interpolated using the found values. Algorithm 2 outlines the Lattice Counting with correction. It applies the NNLS correction step at line 6. Missing O or X values are estimated using the Power-law hypothesis in Section 5.3.

7. EXPERIMENTAL EVALUATION

7.1 Experimental Setup

Data sets: We have conducted experiments using two types of data sets: real-life data and synthetic data. The DBLP data set is used as the real-life data. It is built as described in [2]. The DBLP data set consists of 794,061 sets. We call this full data set the $800K$ data set. Note that this full data set corresponds to more than 300 billion pairs! We also use part of this data set to show scalability. In particular, we use $40K$, $80K$, $160K$, $240K$ and $400K$ randomly selected subsets. The average set size is 14, and the smallest is 3 and the biggest is 219. The synthetic data set is generated using the IBM Quest synthetic data generator [20]. The data set contains 50,000 sets and the universe of sets is 10,000. We varied the average set size from 15 to 50. The pattern correlation parameter is set to 0.25.

Algorithms compared: We implemented the following algorithms for the SSJoin size estimation.

- $LC(\xi)$: It is Lattice Counting with the approximation in Section 5.2. ξ gives the minimum support threshold and we use a value between 0.015% and 0.10%. The estimation is corrected with Algorithm 2. The exact LC is not used due to its long pattern mining time.
- $LCNC(\xi)$: It is identical to $LC(\xi)$ except that the error correction procedure, Algorithm 2, is not applied after the initial estimation.
- IS: This estimation does not rely on the IE principle and Lattice Counting; it only relies on Power-law estimation. Given $LC(t)$, it assumes that all signature patterns are independent and, sums F_ℓ's, $\ell \geq t$.
- $HS(\rho)$: This is an adaptation of Hashed Sampling [17] to SSJoin problem. ρ gives the sampling ratio.

A signature size of 10 is used with a hash space of 2^{15}. For LC and $LCNC$, we use the relaxing parameter $k = 2$ as described in Section 5.2. Linear regression for the Power-law parameter estimation needs special attention. In general, the tail of the Power-law distribution is not reliable. We used only up to 40 leftmost points. The max iteration count θ in Algorithm 2 is set to 100. However, the solution of Equation (11) converged without reaching the count limit most of the time. We apply sanity bounds to LC and O values in line 4 of Algorithm 2. Specifically, we make sure $LC(\ell)$ is no smaller than $LC(\ell+1)$ by setting it to $LC(\ell+1)$ if $LC(\ell) < LC(\ell+1)$. Moreover, if $LC(\ell) < 0$, we take the geometric mean of $LC(\ell')$ and $LC(\ell'')$ that are the first positive values such that $\ell' < \ell$ and $\ell'' > \ell$. In the extreme case of $O(\ell) = 0$ for $1 \leq \ell \leq M$, we set $O(M)$ to 1. For the SSJoin algorithm in HS, we used the *ProbeOptMerge* algorithm [29], which can be used with Hashed Sampling and Jaccard similarity. We did not implement the clustering version since the the improvement was marginal [29, 3].

Evaluation metric: We use absolute count and average (absolute) relative error to show the accuracy. A relative error is defined as $|est\_size - true\_size|/|true\_size|$. We use both measures since the average relative error favors underestimation and can be misleading. That is even if we always answer with 0, the error is capped by -100%. Counts are shown in log scale as the distribution is highly skewed. Given the nature of random sampling, accuracy figures are given with the median value across three runs. This applies to HS, as well as the subsets of the full DBLP data set.

For efficiency, we measure the runtime, which is divided into *pre-processing* time and *estimation* time. HS assumes pre-complied samples and *pre-processing* time corresponds to the time for building the sampled inverted index. For LC, $LCNC$ and IS, it is the time necessary for Min-Hash signature generation. The *estimation* time is the time required for the actual estimation. For HS, it is the SSJoin time on the samples. For LC, $LCNC$ and IS, the time includes

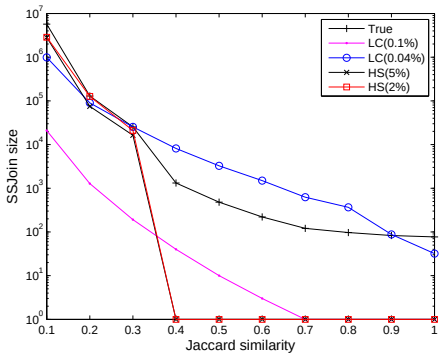
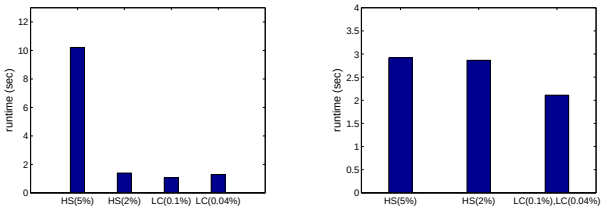


Figure 11: Accuracy on the DBLP data set



(a) Estimation time **(b) Pre-processing time**

Figure 12: Performance on the DBLP data set

the frequent pattern mining time, level sum computation time, as well as the time for Lattice Counting. LC adds the $NNLS$ solving time on top of it. All the times measured include disk I/O's.

We implemented all the estimation algorithms in Java. We downloaded the NNLS solver package from [18]. For the pattern mining algorithm, we downloaded the trie-based Apriori algorithm from [14], but almost any frequent pattern mining algorithms can be used for our estimation purposes. We ran all our experiments on a desktop PC running Linux Kernel 2.6.22 over a 3.00 GHz Pentium 4 CPU with 1GB of main memory.

7.2 DBLP Data set: Accuracy and Efficiency

We first report the results on the accuracy and runtime using the $40K$ DBLP data set. Figure 11 shows the true and estimated SSJoin sizes of $LC(0.1\%)$, $LC(0.04\%)$, $HS(5\%)$, and $HS(2\%)$. $HS(5\%)$ and $HS(2\%)$ deliver relatively accurate estimates for $\tau \leq 0.3$. However, we may be more interested in higher threshold ranges. For $\tau \geq 0.4$, both $HS(5\%)$ and $HS(2\%)$ consistently estimate the join size to be zero, which is not a meaningful estimate. $LC(0.1\%)$ performs better than $HS(5\%)$ and $HS(2\%)$ for τ between 0.4 and 0.7. However, the clear winner is $LC(0.04\%)$ which is by far the closest to the true size.

Figure 12 shows the runtime performance. It is clear that for both estimation time and pre-processing time, LC is at least as efficient as HS. Yet, as shown in Figure 11 and discussed before, LC can be significantly more accurate, particularly for $\tau \geq 0.4$.

7.3 Effectiveness of Lattice Counting and Error Correction

Figure 13 (a) shows the average relative error of IS, $LC(0.02\%)$ and $LCNC(0.02\%)$ in the $160K$ DBLP data set. Recall that IS does not apply Lattice Counting and $LCNC$ does not apply the error correction procedure in Section 6. A huge overestimation is observed in IS. $LCNC$ is better than IS but still shows a rather large overestimation. LC shows the best accuracy. Its errors are very small especially in the high threshold range. This verifies that the Lattice Counting effectively considers the overlaps in counting and that the error correction step indeed offsets the systematic overestimation by Min-Hashing.

Figure 13 (b) shows the results of the full data set. An interesting difference is that IS almost consistently produces a relative error of -1. At first glance, -1 might seem better than $LCNC$; but this indicates that the IS estimation is zero and is not useful. The underestimation is due to the poor estimation of Power-law parameters. As stated in Section 5.3, we use the Power-law distribution of number of pairs-similarity relationship. In this case, IS performs poor parameter estimation, i.e., an overestimation of α, and this results in the severe underestimations. We can see IS produces very unstable estimation. The overestimation of $LCNC$ is again apparent and we can see that the error correction step is very effective.

Figure 13 (c) shows of the estimation time for each algorithm using the 160K DBLP data set. This graph clearly shows that lattice counting (Section 4), the Power-law parameter estimation (Section 5) and overestimation correction (Section 6) all take negligible runtime overhead. The NNLS optimization generally takes less than 50 milliseconds.

7.4 Scalability

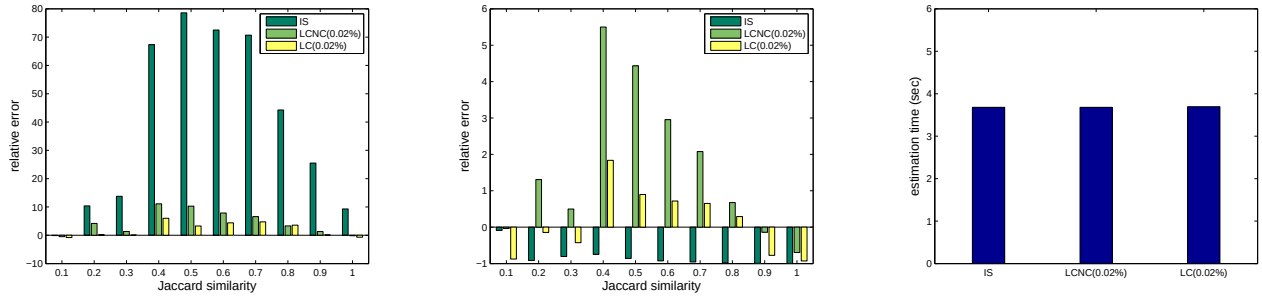
Figure 14 compares the accuracy and runtime of $LC(0.02\%)$ and $HS(1\%)$ varying the data set size with $\tau = 0.8$. $HS(1\%)$ does not provide meaningful estimates until the data set size reaches $400K$ where there are sufficiently many highly similar pairs. Its produces zero estimates for size between $40K$ and $160K$.

The estimates of $LC(0.02\%)$ are quite reliable being consistently below a relative error of 10. A relative error of 10 may seem big; but it is reasonable considering the high selectivity of a high threshold such as $\tau = 0.8$. For instance, in the $80K$ data set, there are more than 6 billion pairs and the selectivity of $\tau = 0.8$ is about 0.000012% compared to the total number of pairs.

Figure 14 (b) and (c) show the pre-processing and estimation time of $HS(1\%)$ and $LC(0.02\%)$. $LC(0.02\%)$ has consistently lower pre-processing time than $HS(1\%)$. We can observe the quadratic increase of estimation time of $HS(1\%)$. $LC(0.02\%)$ also exhibits an increase in the estimation time primarily due to the increase in the frequent pattern mining time. However the increase is rather mild. Chuang et al. proposed a sampling method for computing the pattern distribution [9]. Such a technique should make LC even more scalable.

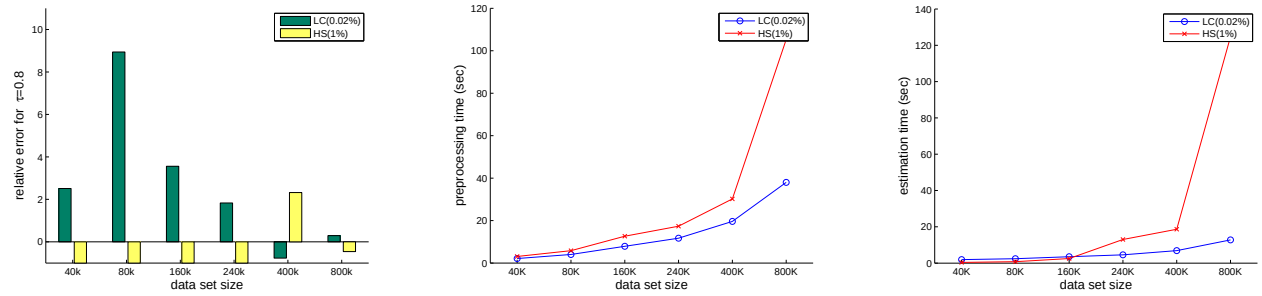
7.5 Synthetic Data set: Accuracy, Efficiency and Scalability

We now report the results on the synthetic data set. We varied the average set size parameter from 15 to 50. Figure 15 (a) and (b) show the accuracy when the average set size is 20 and 25 respectively. In both cases, the estimation of $LC(0.04\%)$ is close to the true SSJoin size. The estimates



(a) 160K (b) 800K (c) Runtime

Figure 13: Effectiveness of Lattice Counting and the error correction



(a) Relative error (b) Pre-processing time (c) Estimation time

Figure 14: Scalability using the DBLP data set

generated by $LC(0.1\%)$ are not as good as those generated by $LC(0.04\%)$. However, the estimation is still close to the true size and and generates reasonable estimates. Both $HS(5\%)$ and $HS(10\%)$ produce reliable estimates at lower similarity ranges, but fail to create non-trivial estimates at high simialrity ranges.

The following table shows how the estimation time (in seconds) of $LC(0.1\%)$ and $HS(5\%)$ varies with average set size. LC is not much affected by increasing the average set size. LC performs analysis on the standardized representation of sets, Min-Hash signatures. Thus, its runtime is almost invariant to the original set size. In contrast, $HS(5\%)$ is affected greatly by increasing the average set size. This is due primarily to the size of the inverted index.

avg. set size	15	20	25	30	50
LC(0.1%)	1.34	1.39	1.39	1.38	1.34
HS(5%)	0.71	1.17	1.77	2.47	6.43

7.6 On Power-law Hypotheses

In this paper, the key hypothesis is that using sampling to estimate SSJoin size is unreliable. A better approach is to rely on Power-laws and to estimate the parameters of Power-laws in a sample-independent fashion. Certainly, one can argue that the minimum support threshold is effectively doing "subsetting". But we believe that subsetting by the minimum support threshold is a more principled way of selection than sampling is. The experimental results shown so far support this hypothesis.

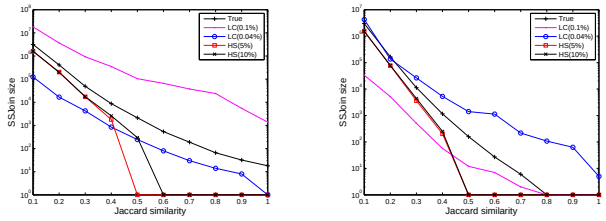
An interesting question would be if the Power-law really holds in the data set. In Section 5, we rely on two Power-law hypotheses. The first one is on the count-frequency relationship in the patterns and the second one is on the number of pairs-similarity relationship. Proving that a distribution really follows a Power-law distribution is not a trivial task. Some of the efforts to quantify the relationship includes the p-value for the Kolmogorov-Smirnov test [10]. However, in our case, the number of points are not sufficient for drawing a conclusion with statistical significance. Moreover, an approximate modeling was enough for our purposes. Therefore we show the log-log plots instead. Figure 7 and Figure 16 present the number of pairs distribution-similarity distribution and the pattern count-frequency distribution. Our observations agree with the findings in [9], but formally verifying the relationship is an open problem.

8. CONCLUSION

We propose an accurate and efficient technique for the SSJoin size estimation. Our technique uses Min-Hash signatures which are succinct representation of sets. We propose a lattice based counting technique called Lattice Counting to efficiently count the number of pairs satisfying the similarity threshold. We exploit Power-law distributions to efficiently compute the pattern distribution necessary for the Lattice Counting. A systematic overestimation by relying on Min-Hash signatures is observed and we also propose a procedure to correct it. In the future, we plan to exploit sampling in the frequent pattern mining for the pattern distribution [4, 9].

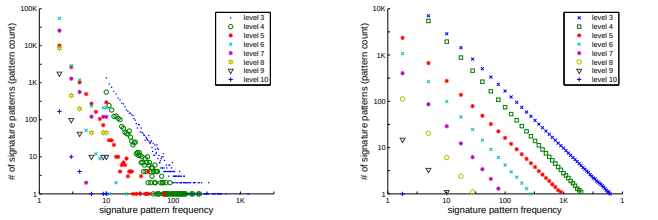
9. REFERENCES

[1] R. Agrawal and R. Srikant. Fast Algorithms for



(a) Avg set size=20 **(b) Avg set size=25**

Figure 15: Accuracy on the synthetic data set



(a) DBLP:pattern count-freq **(b) Syn.:pattern count-freq**

Figure 16: Power-law hypotheses

Mining Association Rules. In *Proc. VLDB*, pages 487–499, 1994.

[2] A. Arasu, V. Ganti, and R. Kaushik. Efficient Exact Set-Similarity Joins. In *Proc. VLDB*, pages 918–929, 2006.

[3] R. J. Bayardo, Y. Ma, and R. Srikant. Scaling up all pairs similarity search. In *Proc. WWW*, pages 131–140, 2007.

[4] M. Boley and H. Grosskreutz. A Randomized Approach for Approximating the Number of Frequent Sets. In *Proc. ICDM*, pages 43–52, 2008.

[5] A. Z. Broder. On the Resemblance and Containment of Documents. In *Proc. SEQUENCES*, pages 21–29, 1997.

[6] A. Ceglar and J. F. Roddick. Association mining. *ACM Computing Surveys*, 38(2), 2006.

[7] S. Chaudhuri, V. Ganti, and R. Kaushik. A Primitive Operator for Similarity Joins in Data Cleaning. In *Proc. ICDE*, page 5, 2006.

[8] Z. Chen, F. Korn, N. Koudas, and S. Muthukrishnan. Selectivity Estimation For Boolean Queries. In *Proc. PODS*, pages 216–225, 2000.

[9] K.-T. Chuang, J.-L. Huang, and M.-S. Chen. Power-law relationship and self-similarity in the itemset support distribution: analysis and applications. *The VLDB Journal*, 17(5):1121–1141, 2008.

[10] A. Clauset, C. R. Shalizi, and M. E. J. Newman. Power-law distributions in empirical data. *SIAM Review*, (to appear) 2009.

[11] E. Cohen. Size-estimation framework with applications to transitive closure and reachability. *Journal of Computer and System Sciences*, 55(3):441–453, 1997.

[12] T. Dasu, T. Johnson, S. Muthukrishnan, and V. Shkapenyuk. Mining database structure; or, how to build a data quality browser. In *Proc. SIGMOD*, pages 240–251, 2002.

[13] C. Faloutsos, B. Seeger, A. Traina, and J. Caetano Traina. Spatial join selectivity using power laws. In *SIGMOD*, pages 177–188, 2000.

[14] Ferenc Bodon. Apriori implementation of ferenc bodon. http://www.cs.bme.hu/ bodon/en/apriori/.

[15] A. Fuxman, E. Fazli, and R. J. Miller. Conquer: Efficient management of inconsistent databases. In *Proc. SIGMOD*, pages 155–166, 2005.

[16] M. Hadjieleftheriou, A. Chandel, N. Koudas, and D. Srivastava. Fast Indexes and Algorithms for Set Similarity Selection Queries. In *Proc. ICDE*, pages 267–276, 2008.

[17] M. Hadjieleftheriou, X. Yu, N. Koudas, and D. Srivastava. Hashed Samples: Selectivity Estimators For Set Similarity Selection Queries. In *Proc. VLDB*, pages 201–212, 2008.

[18] Hans D. Mittelmann. Decision tree for optimization software. http://plato.asu.edu/sub/nonlsq.html.

[19] M. A. Herna'ndez and S. J. Stolfo. The merge/purge problem for large databases. In *Proc. SIGMOD*, pages 127–138, 1995.

[20] IBM. Quest synthetic data generation code. http://www.almaden.ibm.com/cs/disciplines/iis/.

[21] P. G. Ipeirotis, E. Agichtein, P. Jain, and L. Gravano. To search or to crawl?: towards a query optimizer for text-centric tasks. In *Proc. SIGMOD*, pages 265–276, 2006.

[22] R. Jin, S. McCallen, Y. Breitbart, D. Fuhry, and D. Wang. Estimating the Number of Frequent Itemsets in a Large Database. In *Proc. EDBT*, pages 505–516, 2009.

[23] C. L. Lawson and R. J. Hanson. *Solving Least Squares Problems*. Society for Industrial Mathematics, 1987.

[24] H. Lee, R. T. Ng, and K. Shim. Extending Q-Grams to Estimate Selectivity of String Matching with Edit Distance. In *Proc. VLDB*, pages 195–206, 2007.

[25] H. Lee, R. T. Ng, and K. Shim. Approximate Substring Selectivity Estimation. In *Proc. EDBT*, pages 827–838, 2009.

[26] A. Metwally, D. Agrawal, and E. E. Abbadi. DETECTIVES: DETEcting Coalition hiT Inflation attacks in adVertising nEtworks Streams. In *Proc. WWW*, pages 241–250, 2007.

[27] T. Pitoura and P. Triantafillou. Self-join size estimation in large-scale distributed data systems. In *ICDE*, pages 764–773, 2008.

[28] M. Sahami and T. Heilman. A Web-based Kernel Function for Measuring the Similarity of Short Text Snippets. In *Proc. WWW*, pages 377–386, 2006.

[29] S. Sarawagi and A. Kirpal. Efficient set joins on similarity predicates. In *Proc. SIGMOD*, pages 743–754, 2004.

[30] G. K. Zipf. *Human Behavior and the Principle of Least Effort*. Addison-Wesley, 1947.