$\mathcal{O}$-Bin : Oblivious Binning for Encrypted Data over Cloud

Mahmood Ahmad *, Zeeshan Pervez †, Byeong Ho Kang,‡, Sungyoung Lee *
*Department of Computer Engineering, Kyung Hee University,South Korea
†School of Computing, University of the West of Scotland, Paisley, PA1 2BE, United Kingdom
‡School of Computing and Information Systems, University of Tasmania, Australia
Email: rayemahmood *, zeeshan.pervez†, Yiyoon‡, Byeong.Kang‡,sylee*@ *oslab.khu.ac.kr,†uws.ac.uk,

***Abstract*—In recent years, the data growth rate has been observed growing at a staggering rate. Considering data search as a primitive operation and to optimize this process on large volume of data, various solution have been evolved over a period of time. Other than finding the precise similarity,these algorithms aim to find the approximate similarities and arrange them into bins. Locality sensitive hashing *(LSH)* is one such algorithm that discovers probable similarities prior calculating the exact similarity thus enhance the overall search process in high dimensional search space. Realizing same strategy for encrypted data and that too in public cloud introduces few challenges to be resolved before probable similarity discovery. To address these issues and to formalize a similar strategy like LSH, in this paper we have formalized a technique $\mathcal{O}$-Bin that is designed to work over encrypted data in cloud. By exploiting existing cryptographic primitives, $\mathcal{O}$-Bin preserves the data privacy during the similarity discovery for the binning process. Our experimental evaluation for $\mathcal{O}$-Bin produces results similar to LSH for encrypted data.**

***Keywords*-Similarity discovery; Security and Privacy; Cloud; Binning**

I. Introduction

In recent years the data growth rate is in fast gear with its projection estimated to grow 44 times within a time period of 11 years, $2009 - 2020$ [1]. This phenomenon of data explosion necessitates efficient data exploration methods and techniques so that applications that are build over large volume of data can glean insight in least amount of time. Considering search as a primitive operation for data exploration, certain techniques like binary trees, k-d trees and hash tables are used for fast lookups. As a next step and to further enhance searching mechanism, a technique known as locality sensitive hashing (LSH) has been proven valuable for retrieving items that are similar to a search criterion in a high dimensional search space [2]. This approach belongs to a novel and interesting class of algorithms that are known as randomized algorithms. For very large database system with high dimensions, LSH drastically reduces the computational time at the cost of small probability of failing to find the absolute closet match. For given user search, LSH prepares most probable and potential subset form entire data and organize them into bins [3]. The data set sharing similar bin have the high probability of similarity in between them. Using an example shown in Figure 1 we will explain how binning is useful for efficient search. In this example the objects a, b, c, and d have certain attributes and using their common attributes these objects can be placed in similar bins. The Object *a* and *b* have similarity with respect to their geometrical shape which we denote as *shape(a,b)*. Similarity other similarities that exist amongst other objects are *color(a,c)*, *toughness(c,d)*, and *weight(d,b)*. If a certain search criterion $\mathcal{C}$ is need to be evaluated on this pre-processed binning then only a subset of dataset will be searched within, instead of iterating through the whole.

Figure 1. Example:Finite set world space

With the staggering growth of data and to maximize its utility impact, data outsourcing in public cloud is gaining momentum. Besides data outsourcing, services that are built over this data are also executed within cloud infrastructure. Considering services similar to LSH, their execution in cloud infrastructure can help cloud server in creating bins of similar objects. However; if outsourced data is sensitive in nature then additional requirements need careful considerations before the binning process start, these are

- for data privacy and security, encrypted outsourcing is strongly recommended [4],
- the attributes of encrypted data need to be compared in such way that it should not reveal any information about the outsourced data i.e.,working with the encrypted data directly,
- while working with the encrypted attributes, the binning process must resemble like LSH are close to it.

Considering these challenges and motivated by the LSH scheme, in this paper we have formalized a similar approach that is aimed to optimize search processing over encrypted data by binning similar objects probabilistically.

By exploiting the existing cryptographic primitives our scheme $\mathcal{O}$-Bin is capable of working with the encrypted attributes directly, thus hiding the internal details of actual data. Due to this characteristic, it is feasible to deploy it in an untrusted domain of public cloud to leverage its computational resources obliviously. To analyze the performance of our model in comparison to the LSH scheme we report our experimental evaluation that is conducted on similar data set.The rest of the paper is organized as follows.

Section II is about the definitions and preliminaries for techniques that are used during the construction of $\mathcal{O}$-Bin. Assumptions and notations used for the descriptive detail of $\mathcal{O}$-Bin is given in Section III. General overview of LSH with respect to $\mathcal{O}$-Bin is given in Section IV. System execution, construction of $\mathcal{O}$-Bin in cloud server and detailed explanation of $\mathcal{O}$-Bin with real data is given in Section V. Section VI is on Related work. The paper concludes in Section VII.

II. Definitions

A. Homomorphic Encryption

Homomorphic encryption *HE* is a form of encryption where a specific algebraic operation performed on the plaintext is equivalent to another (possibly different) algebraic operation performed on the ciphertext. An encryption scheme is said to be additive homomorphic if and only if

$$E_H(m_1) \odot E_H(m_2) = E_H(m_1 + m_2)$$

where $\odot$ is an operator. Pascal Paillier cryptosystem [5] possesses the property of additive *HE* which is as follows.

- Key generation: Let $N = pq$ be the RSA-modulus and g be an integer of order αN module N^2 for some integer α. The public key is (N, g) and the private key is $\lambda(N) = lcm((p-1)(q-1))$.
- Encryption: The encryption of message $m \in Z_N$ is $E_h(m) = g^m r^N \ mod\ N^2$ where $r \in_R Z_N^*$
- Decryption: For ciphertext c, the message is computed from

$$m = \frac{L(c^{\lambda(N)}\ mod\ N^2)}{L(g^{\lambda(N)}\ mod\ N^2)}$$

A scheme is said to be multiplicative homomorphic if and only if

$$E_H(m_1) \odot E_H(m_2) = E_H(m_1 \times m_2)$$

The Goldwasser-Micali (GM) cryptosystem is a semantically-secure scheme based on the quadratic residuosity problem. It has XOR homomorphic properties, in the sense that $E_H(b).E_H(b') = E(b \oplus b')\ mod\ N$ where b and b' are bits and N is the public key. A homomorphic encryption is said to be semantically secure if $E(H)$ reveals no information about m_1 and m_2, hence it is computationally infeasible to distinguish between the cases $m_1 = m_2$ and $m_1 \neq m_2$ [6].

B. Private Matching

Yaos classical millionaires problem involves two millionaires who wish to know who is richer. However, they do not want to find out inadvertently any additional information about each others wealth. More formally, given two input values x and y, which are held as private inputs by two parties Alice and Bob respectively. The problem is to securely evaluate the Greater Than (GT) condition through a predicate function f such that $f(x, y) = 1$ if and only if $x > y$, without exposing inputs. We used Fischlin protocol [7] for the private comparison because it allows comparing two ciphertexts encrypted with the GM crytosystem using the same public key. Fischlin uses the GM-encryption scheme to construct a two-round GT protocol. The GM encryption scheme has the XOR, NOT and re-randomization properties. They modified the scheme to get an AND property, which can be performed only once. The computation cost $O(n)$ for the server side is very efficient. Nevertheless, the overall computation cost for both the client and server sides are $O(nlogN)$, where N is the modulus. The scheme is as follows.

- Key generation: Let $N = pq$ be the RSA-modulus and z be a quardatic non-residue of Z_n^* with Jacobi symbol +1. The public key is (N, z) and the secret key is (p, q)
- Encryption: For a bit b, the encryption is $E(b) = z^r r^2 \bmod N$, where $r \in_R Z_N^*$
- Decryption: For a ciphertext c, its plaintext is 1 if and only if c is a quardatic non-residue. If c is a quadratic residue in Z_N, c is quadratic residue in both Z_p^* and Z_q^*
- xor-property: $E(b_1)E(b_2) = E(b_1 \oplus b_2)$
- not-property: $E(b) \times z = E(b \oplus 1 = E(\bar{b})$
- re-randomization: Randomization of ciphertext c can be done by multiplying an encryption of 0

C. The Bloom Filter

A Bloom Filter is a method for representing a set $S = s_1, s_2, \ldots, s_n$ of keys from a universe U, by using a bit vector V of $m = O(n)$ bits. It was invented by Burton Bloom in 1970 [8]. All the bits in the vector V are initially set to 0. The Bloom Filter uses k hash functions, $h_1, h_2, \ldots, h_k$ mapping keys from U to the range $1 \ldots m$. For each element in $s \in S$, the bits at positions $h_1(s), h_2(s), \ldots, h_k(s)$ in V are set to 1. Given an item $q \in U$, its membership is checked in S by examining the bits at positions $h_1(q), h_2(q), \ldots, h_k(q)$. If one or more of the bits is equal to 0 then q is certainly not in S. Otherwise q is considered as a member of S but with certain false positive rate. This probability depends upon the parameter selection adopted for the Bloom Filter, namely m and k. After inserting n keys at random to the array of size m the probability that a particular bit is 0 or 1 is $(1 - \frac{1}{m})^{kn}$. The error probability for a bloom filter E_b is give in equation

$$E_b = \left(1 - \left(1 - \frac{1}{m}\right)^{kn}\right)^k \approx \left(1 - e^{-\frac{kn}{m}}\right)^k \quad (1)$$

III. Notations and Assumptions

We assume that data ownership has sufficient resources for data encryption and attribute encryption on the relevant index before outsourcing it on the public cloud. While

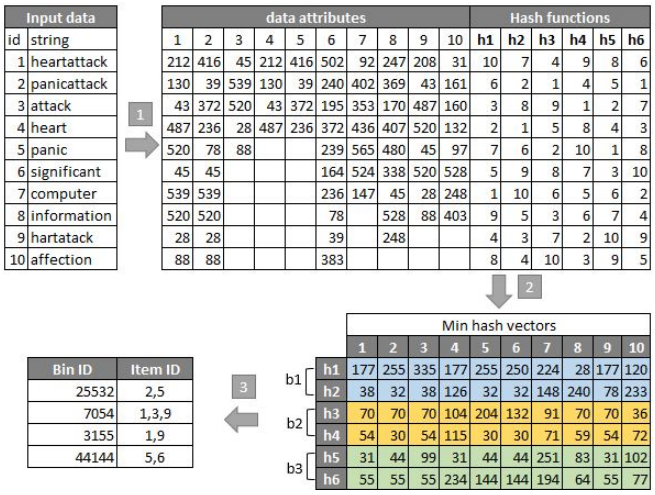

Figure 2. Finding the probable similarity using LSH binning

explaining $\mathcal{O}$-Bin, we only focus on the oblivious binning process and intentionally neglect the details for security key construction and details for their distribution. As a common security assumption we assume that the cloud server is a semi-honest entity [9]. The semi honest entity behaves honestly but try to extract additional information. We also assume that the data instances have been declared with their attributes. The notations used for the descriptive detail of proposed model are given in table I.

Table I
NOTATIONS USED IN THE DESCRIPTIVE DETAIL

Notation	Description
$\mathcal{D}$	Data to be outsourced in public cloud
d	An instance of data i.e., $d_i \in D$
$\mathcal{A}$	Attribute list of d where $\mathcal{A} = \{a_1, a_2, \ldots, a_n\}$
β	Bloom filter representation of d
$\mathcal{H}$	Family of hash functions used for bloom filter construction
λ	Length of bloom filter: total number of bits in β_i that are either set to *zero* or *one*
$\dagger_i$	Total number of bit positions in β_i that are set to *one* i.e.,*true*
$\mathcal{E}_s$, $\mathcal{D}_s$	Symmetric encryption and decryption algorithms
$\mathcal{E}_H$, $\mathcal{D}_H$	Homomorphic encryption and decryption algorithms
σ_{pk}, σ_{sk}	Public and secret key pair for Homomorphic encryption

IV. General Overview and System Formulation

Before we present the proposed model, we start with the bins construction using LSH on a given data set as shown in Figure 2. In validation to our proposed solution, first we apply LSH on this data and note the results. This input data used for LSH is then encrypted and $\mathcal{O}$-Bin results are compared with the LSH results. The input data consist on a finite set of data strings where each string is transformed into attributes for its multidimensional representation in some metric space. To explain this transformation we will use one instance of input data as an example, 'heartattack'. First step in this transformation is to create its n-gram tokens with length 2 such that for an input data having n characters, there will be $(n-1)$ tokens. Hence the tokens will be 'he', 'ea' ..., and 'ck'. The reason for selecting token length=2 is to use its first and second character as i^{th} and j^{th} coordinates respectively. The token t_{ij} is then mapped onto a table shown in Figure 5. In current example scenario, first token t_{he} will be mapped to 212 i.e., 8^{th} row and 5^{th} column and so on. Similar transformation is then applied to all instances of input data and shown in step-1 of Figure 2. We then took six random hash functions $(h_1, h_2, \ldots, h_6)$ and calculate the min hash values that is shown in step-2 of Figure 2. The min hash vectors are then divided into b bands of r rows, where $b = 3$ and $r = 2$ in our current example. In step-3, the LSH will create bins and identify certain ids of input data sharing the similar bins. In Figure 2 we only show those bins having more than one items. According to LSH, the probability of similarity existence is shown between item id $(2, 5)$, $(1, 3, 9)$, $(1, 9)$ and $(5, 6)$. Although LSH does not guarantee an exact answer (see $(5, 6)$) but instead provides a high probability guarantee that it will return the correct answer or one close to it. For large data that is resident in public cloud, it can be realized that LSH can drastically reduce the search cost by its unique methodology. The cloud server can effectively utilize random hash functions over outsourced data to perform LSH operations.

With these results, now we explain how similar operation for the construction of binning for the encrypted data take place in the cloud.

V. System Execution: Construction of Bin on Cloud Server

Using the notations given in table I, $\mathcal{O}$-Bin will execute as follows. The data owner creates a bloom filters $\beta_1, \beta_2, \ldots, \beta_n$ for all $d_i \in \mathcal{D}$, shown in Figure 3. For each bit in bloom filter, it is then encrypted under the secret key of homomorphic encryption σ_{sk}. Due to the probabilistic nature of homomorphic encryption, given two encrypted bits in a bloom filter it would be indistinguishable for an adversary to know that originally the bits are either set as *zero* or *one*. The data files $\mathcal{D}$ are encrypted with the symmetric key and denoted as $\mathcal{D}^{E_s}$. The encrypted bloom filters and encrypted data files are now uploaded on the cloud where cloud will find the approximate similarity satisfying the pre-defined threshold value k. During system initialization, cloud server will also receive σ_{pk} for the homomorphic operations on the encrypted bloom filter. Besides σ_{pk}, cloud server will also receive the similarity threshold value k which is encrypted using the Goldwasser-Micali (GM) cryptosystem [10]. For private comparison, Fischlin protocol [7] allows comparing two ciphertext encrypted with the GM cryptosystem using the same public key. While discovering the probable similarity and their placement into bins, cloud server will use GM cryptosystem along with the Fischlin's protocol for the private comparison only. Prior private comparison, the homomoprhic operations that are performed on the cloud server are given as follows. For two bloom filters, β_i and β_{i+1} a bit by bit homomorphic multiplication is performed to obtain their resultant as $\Delta^{\otimes}$. To calculate the similarity

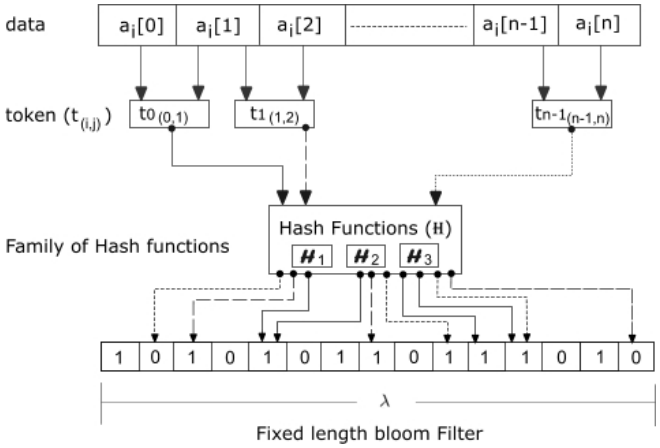

Figure 3. Construction of fixed length bloom filter

score $\Delta^{\oplus}$ for β_i and β_{i+1}, each bit that belongs to $\Delta^{\otimes}$ is then added together according to equation 2.

$$\Delta^{\oplus}_{i,i+1} = \Delta^{\otimes}_{i}[0] \oplus \Delta^{\otimes}_{i}[1] \oplus, \ldots, \Delta^{\otimes}_{i}[\lambda] \qquad (2)$$

If this $\Delta_{i,i+1} \geq k$ then β_i and β_{i+1} are assigned with the similar bin. Likewise, the cloud server will evaluate all the entries and finalize the binning process. During this oblivious process the only information cloud learns is the cardinality of each bin.

A. $\mathcal{O}$-Bin with real data

To build our system we use the bloom filters that is applied on the $n-grams$ of input data. For each instance of input data, its $n-grams$ are used as input feed into the bloom filter as shown in the figure 3. The output of this operation is a fixed length bloom filter β of length λ, where each bit of β is set as *zero* or *one*. The information that we use from bloom filter for binning purpose is the number of bits that are set to *one* or *"true"* i.e., †. By using this information we create bins in three possible ways as shown in figure 4. First, the bins are created according to similar number of bits set to *true*. This arrangement makes a total of 8 bins where number of true bits are shown in grey circle, Figure 4-(a). In this arrangement only $B7$ with $\dagger = 21$ holds more than one data instances. This arrangement is far below in comparison with what LSH provides. Therefore; instead of relying on similar number of true bits, we expand true bit comparison as $\dagger \pm 2$. If we look at $B4$ in Figure 4-(b), now there are two data instances in this bucket instead of one. This arrangement improves the binning candidacy but still it falls below than desirable results of probable similarity discovery. Figure 4-(c) shows the final and probable arrangement of data instances (shown with respect to their ids) in each bin. In third arrangement the proposed model creates only 5 bins and participation of data instances in each bin is more close to what LSH provides. In Figure 4-(c) we show the id of each bin and id of data instance only. The explanation for constructing such binning arrangement is as follows.

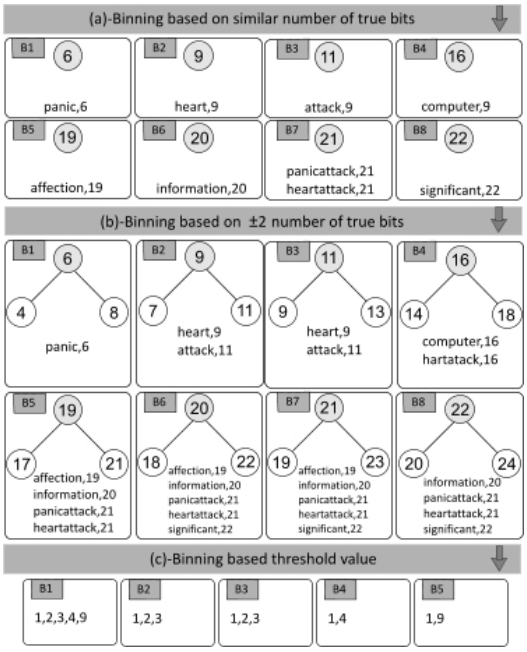

Figure 4. Creation of $\mathcal{O}$-Bin

To achieve results that are more close with LSH, we define a matching threshold value k for any two bloom filters β_1 and β_2. For our current experimental setup, we set $k = 60$, which means that for any β_n bloom filter to become a member of bin B_i, it must satisfy similarity $\geq k$ with any of β_{n-1} that already have membership in B_i i.e., $\beta_{n-1} \in B_i$. If we look at data instance with id $1, 3$, and 9 it can be observed that they share certain characters (attributes in this case) in common. If two data instances having bloom filtered representation as $B_i = [1, 1, 0, 1, 0, 1, 0, 1]$ and $B_{i+1} = [1, 1, 0, 0, 0, 1, 0, 1]$, then according to equation 1 and with threshold similarity $k \geq 60$ both will be in similar bin.

VI. Related Work

To protect the privacy of distributed sources using cryptographic techniques was first applied in the area of data mining for the construction of decision trees by Lindell and Pinkas[11]. This work falls under the framework of secure multiparty computation [12] to achieving "perfect" privacy. The necessity of finding similarity between two entities involves various methodologies. The end goal of all similarity discover solutions is to find nearest matches in least amount of time. The desire to achieve this goal becomes complex when the data in volume is very large and has sizeable number of attributes. To speed up this discovery process, if certain pre-processing is done in such a way that the target search space becomes a subset of entire search space. For example the Web contains many duplicate pages, partly because content is duplicated across sites and partly because there is more than one URL that points to the same content. A solution to identify Web page duplicates makes use of shingles. Each shingle represents a portion of a Web page and is computed by

forming a histogram of the words found within that portion of the page. AltaVista, the first large-scale Web search engine, used random selections (similarly to LSH) to test the similarity of pages [13]. Like wise Shakhnarovich et al [14] uses it for fast image retrieval as object recognition. In music retrieval typically usually conventional hashes and robust features are used to find musical matches. The features can be fingerprints, i.e., representations of the audio signal that are robust to common types of abuse that are performed to audio before it reaches our ears [11].

To achieve this, LSH The information on the world wide found on one web page may contain similar informaiton on the world wide web is duplicated

VII. Conclusion

In this paper we have proposed an oblivious similarity discovery model for encrypted data in cloud environment. Other than encrypted outsourcing, the privacy is also deemed necessary during the processing that take place on the encrypted data. Similar to locality sensitive hashing (LSH), our system $\mathcal{O}$-Bin finds the probable similarities between the participating data objects and assign them with relevant bins. Considering the trend for encrypted outsourcing and search as a primitive operation for data exploration, $\mathcal{O}$-Bin provides efficient searching mechanism like LSH.

VIII. Acknowledgments

This work is supported by the Industrial Core Technology Development Program (10049079 , Development of Mining core technology exploiting personal big data) funded by the Ministry of Trade, Industry and Energy (MOTIE, Korea)

References

[1] J. Gantz and D. Reinsel, "The digital universe in 2020: Big data, bigger digital shadows, and biggest growth in the far east," *IDC iView: IDC Analyze the Future*, 2012.

[2] A. Rajaraman and J. D. Ullman, *Mining of massive datasets*. Cambridge University Press, 2011.

[3] M. Slaney and M. Casey, "Locality-sensitive hashing for finding nearest neighbors [lecture notes]," *Signal Processing Magazine, IEEE*, vol. 25, no. 2, pp. 128–131, 2008.

[4] R. Dingledine and P. Golle, *Financial Cryptography and Data Security*. Springer, 2009, vol. 5628.

[5] H.-Y. Lin and W.-G. Tzeng, "An efficient solution to the millionaires problem based on homomorphic encryption," in *Applied Cryptography and Network Security*. Springer, 2005, pp. 456–466.

[6] P. Paillier, "Trapdooring discrete logarithms on elliptic curves over rings," in *Advances in CryptologyASIACRYPT 2000*. Springer, 2000, pp. 573–584.

[7] M. Fischlin, "A cost-effective pay-per-multiplication comparison method for millionaires," in *Topics in CryptologyCT-RSA 2001*. Springer, 2001, pp. 457–471.

[8] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Communications of the ACM*, vol. 13, no. 7, pp. 422–426, 1970.

[9] W. Jiang and C. Clifton, "A secure distributed framework for achieving k-anonymity," *The VLDB JournalThe International Journal on Very Large Data Bases*, vol. 15, no. 4, pp. 316–333, 2006.

[10] J. Katz and M. Yung, "Threshold cryptosystems based on factoring," in *Advances in CryptologyASIACRYPT 2002*. Springer, 2002, pp. 192–205.

[11] P. Cano, E. Batle, T. Kalker, and J. Haitsma, "A review of algorithms for audio fingerprinting," in *Multimedia Signal Processing, 2002 IEEE Workshop on*. IEEE, 2002, pp. 169–173.

[12] R. Canetti, "Universally composable security: A new paradigm for cryptographic protocols," in *Foundations of Computer Science, 2001. Proceedings. 42nd IEEE Symposium on*. IEEE, 2001, pp. 136–145.

[13] A. Z. Broder, S. C. Glassman, M. S. Manasse, and G. Zweig, "Syntactic clustering of the web," *Computer Networks and ISDN Systems*, vol. 29, no. 8, pp. 1157–1166, 1997.

[14] G. Shakhnarovich, P. Viola, and T. Darrell, "Fast pose estimation with parameter-sensitive hashing," in *Computer Vision, 2003. Proceedings. Ninth IEEE International Conference on*. IEEE, 2003, pp. 750–757.

		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
		a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
0	a	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51
1	b	52	53	54	55	56	57	58	59	60	61	62	63	64	65	66	67	68	69	70	71	72	73	74	75	76	77
2	c	78	79	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	96	97	98	99	100	101	102	103
3	d	104	105	106	107	108	109	110	111	112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127	128	129
4	e	130	131	132	133	134	135	136	137	138	139	140	141	142	143	144	145	146	147	148	149	150	151	152	153	154	155
5	f	156	157	158	159	160	161	162	163	164	165	166	167	168	169	170	171	172	173	174	175	176	177	178	179	180	181
6	g	182	183	184	185	186	187	188	189	190	191	192	193	194	195	196	197	198	199	200	201	202	203	204	205	206	207
7	h	208	209	210	211	212	213	214	215	216	217	218	219	220	221	222	223	224	225	226	227	228	229	230	231	232	233
8	i	234	235	236	237	238	239	240	241	242	243	244	245	246	247	248	249	250	251	252	253	254	255	256	257	258	259
9	j	260	261	262	263	264	265	266	267	268	269	270	271	272	273	274	275	276	277	278	279	280	281	282	283	284	285
10	k	286	287	288	289	290	291	292	293	294	295	296	297	298	299	300	301	302	303	304	305	306	307	308	309	310	311
11	l	312	313	314	315	316	317	318	319	320	321	322	323	324	325	326	327	328	329	330	331	332	333	334	335	336	337
12	m	338	339	340	341	342	343	344	345	346	347	348	349	350	351	352	353	354	355	356	357	358	359	360	361	362	363
13	n	364	365	366	367	368	369	370	371	372	373	374	375	376	377	378	379	380	381	382	383	384	385	386	387	388	389
14	o	390	391	392	393	394	395	396	397	398	399	400	401	402	403	404	405	406	407	408	409	410	411	412	413	414	415
15	p	416	417	418	419	420	421	422	423	424	425	426	427	428	429	430	431	432	433	434	435	436	437	438	439	440	441
16	q	442	443	444	445	446	447	448	449	450	451	452	453	454	455	456	457	458	459	460	461	462	463	464	465	466	467
17	r	468	469	470	471	472	473	474	475	476	477	478	479	480	481	482	483	484	485	486	487	488	489	490	491	492	493
18	s	494	495	496	497	498	499	500	501	502	503	504	505	506	507	508	509	510	511	512	513	514	515	516	517	518	519
19	t	520	521	522	523	524	525	526	527	528	529	530	531	532	533	534	535	536	537	538	539	540	541	542	543	544	545
20	u	546	547	548	549	550	551	552	553	554	555	556	557	558	559	560	561	562	563	564	565	566	567	568	569	570	571
21	v	572	573	574	575	576	577	578	579	580	581	582	583	584	585	586	587	588	589	590	591	592	593	594	595	596	597
22	w	598	599	600	601	602	603	604	605	606	607	608	609	610	611	612	613	614	615	616	617	618	619	620	621	622	623
23	x	624	625	626	627	628	629	630	631	632	633	634	635	636	637	638	639	640	641	642	643	644	645	646	647	648	649
24	y	650	651	652	653	654	655	656	657	658	659	660	661	662	663	664	665	666	667	668	669	670	671	672	673	674	675
25	z	676	677	678	679	680	681	682	683	684	685	686	687	688	689	690	691	692	693	694	695	696	697	698	699	700	701

Figure 5. Attribute value matrix used for LSH binning