An efficient technique for searching very large files with fuzzy criteria using the Pigeonhole Principle

Maryam Yammahi *Kamran kowsari* *Chen Shen* *Simon Berkovich*

Department of Computer Science, George Washington University
Washington DC, USA
myammahi@gwmail.gwu.edu

***Abstract*—Big Data is the new term of the exponential growth of data in the Internet. The importance of Big Data is not about how large it is, but about what information you can get from analyzing these data. Such analysis would help many businesses on making smarter decisions, and provide time and cost reduction. Therefore, to make such analysis, you will definitely need to search the large files on Big Data. Big Data is such a construction where sequential search is prohibitively inefficient, in terms of time and energy. Therefore, any new technique that allows very efficient search in very large files is highly demanded. The paper presents an innovative approach for efficient searching with fuzzy criteria in very large information systems (Big Data). Organization of efficient access to a large amount of information by an "approximate" or "fuzzy" indication is a rather complicated Computer Science problem. Usually, the solution of this problem relies on a brute force approach, which results in sequential look-up of the file. In many cases, this substantially undermines system performance. The suggested technique in this paper uses different approach based on the Pigeonhole Principle. It searches binary strings that match the given request approximately. It substantially reduces the sequential search operations and works extremely efficiently from several orders of magnitude including speed, cost and energy. This paper presents a complex developed scheme for the suggested approach using a new data structure, called FuzzyFind Dictionary. The developed scheme provides more accuracy than the basic utilization of the suggested method. It also, works much faster than the sequential search.**

Keywords—Big Data; Approximate search; Informationan Retrivel; Pigeonhole Principle; Algorithms and Data Structure.

I. INTRODUCTION

An important operation in the information retrieval processing is to retrieve items with approximately rather than exactly matching attributes. This issue has received a lot of consideration, in a view of the fact that several applications require approximate matching operations. Typically, these applications, such as information retrieval, pattern recognition, computational biology and others [1]. In this kind of matching, we are looking for the closest solution, which depends on the considered type of errors. Considering the information items are represented as binary vector with bit position values featuring the presence or absence of corresponding attributes [2]. Then, it is important to select vectors that are close to the given vector in term of the Hamming distance, which indicates the number of mismatches between two strings of equal length. There are many suggested algorithms in the literature review that deals with the approximate matching problem. However, most of these studies only considered substring matching of a pattern without considering approximate or fuzzy searching. These papers such as [3][4] and [5]. Other studies are based on some sophisticated combinations of approximate matching like [6] and [7].

Achieving an efficient approximate search has become very challenging with the huge availability of the information in the Internet. The challenging in the solution of this problem is that a non-exact access has to be formulated by means of the exact operational instructions [8]. Usually the solution, in the case of having high dimensional objects with a large number of attributes, is based on brute force approach which implies sequential comparison of all the elements of the system with the given information item [2]. In many cases, this substantially undermines system performance. It also, consumes a lot of time and energy. The good new is that the sequential processes can be easily parallelized; however in a very large information system, this would be very expensive and costly solution. Therefore, a fast algorithm that solves this problem is highly demanded. We are presenting a novel searching technique that searches very large files (beyond terabytes) with fuzzy criteria. The basic utilization of this technique has been introduced in [9] and in this paper we are developing a new scheme of the suggested approach based on FuzzyFind method [10][11]. Further details of the suggested method and its development will be explained in the sections below.

II. THE PROPOSED APPROACH (PIGEONHOLE SEARCH)

Organization of the retrieval of information items according to a fuzzy criterion essentially depends on the representation of information items and the method of their comparison [8]. The suggested technique presents efficient searching with fuzzy criteria in very large information systems. It is specifically aimed at selecting Binary Feature Vectors from the rows of Bit-Attribute Matrices that are within a specified Hamming distance. Regularly, this can be done by a sequential lookup. This sequential lookup can be speed up by using Bit-Attribute Matrices in a vertical format. Yet this may be still be not

978-1-4799-4321-0/14 $31.00 © 2014 IEEE
DOI 10.1109/COM.Geo.2014.8

sufficient enough for the case of very large databases, like those beyond terabytes. Fig.1 represents the basic definition of the problem, where we are looking for a Bit-Attribute Vector (BAV) of length m, into a Bit-Attribute Matrix (BAM) that are within a particular Hamming distance. The Bit-Attribute Matrix contains N items form Item [0] to Item [N-1], and Each item is characterized with m attributes from A0 to Am-1, as shown below.

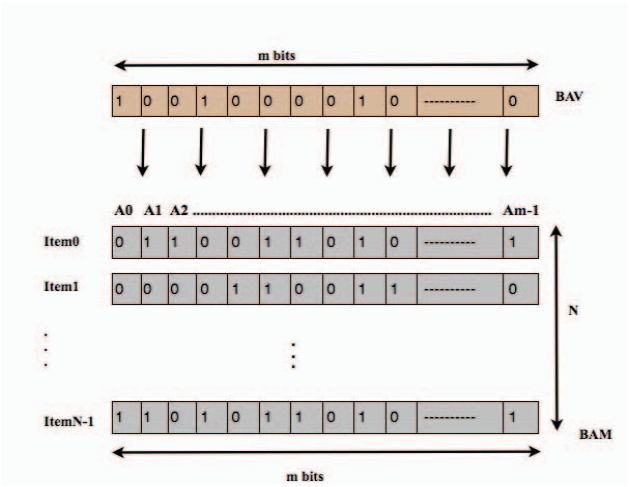

Fig. 1. The basic problem definition

The suggested technique, as we called it "Pigeonhole search", uses different approach based on Pigeonhole Principle. Thus, searching would go much faster. Pigeonhole Principle, states that if we have (n+1) or more pigeons and n holes, then at least two pigeons must be in the same hole [12]. Accordingly, if we have k segments and (k-1) mismatches; then there must be at least one segment with no mismatches (exact match) or less number of mismatches than the other segments. Therefore, we will locate the segments fast if they are within Hamming distance (0) from a given searching pattern. For each such match in each segment, we perform sequential lookup and compare the remaining segments within a specified threshold of the mismatches. Thus, if (μ) is the least number of mismatches that a segment can have, therefore, the threshold of the mismatches would equal to (1). As a result, we will find all rows of the Bit-Attribute Matrix within this distance.

$$\text{Threshold}(\text{d}) = [\text{k} \times (\mu + 1)] - 1 \qquad (1)$$

Therefore, we will partition the Bit-Attribute Vector into (k) segments, each of length (m/k). Then, we will index the Bit-Attribute Matrix by first, partitioning it vertically into k parts. Furthermore, will have organized directory's tables to access corresponding segments of the Bit-Attribute Vector in the Bit-Attribute Matrix. Therefore, we will have k tables, each of length ($2^{m\backslash k}$) vertically. Fig.2 shows the indexing of 16-bit segment. In this scheme we have 64 bits attribute vector's length and has been divided into four segments each of length 16 bits. Each segment will be accessed through its corresponding table to find its location. Thus, the union of all the locations of all the segments will be the candidate list, where, it will be searched sequentially to get all the vectors that are less than or equal to a specific threshold. In the case of the 64-bit vector, the threshold equals to 3 mismatches, where μ is (0). The preliminary step of Pigeonhole search has been described in [9].

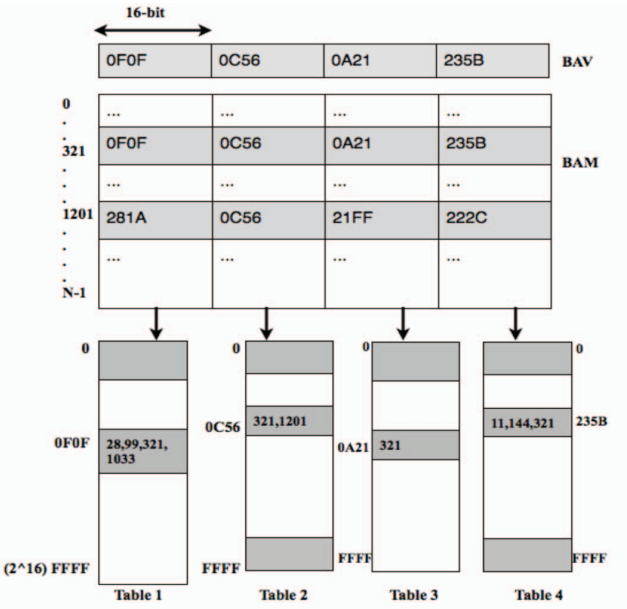

Fig. 2. 16-bit indexing

III. Fault Tolerant Indexing with Hash Transformation of Golay Code

Fuzzy search processes can take advantage from hashing techniques, in order to have fast access. It would be even better, if it were possible to have a hash function that is not sensitive to certain deviations from a standard representation of information items. In hashing, the place where the aimed information items are stored is accessed directly, then the choice of an suitable item requires traversing a relatively small amount of additional data [8]. For example, using SOUNDEX encoding [13], as a preliminary step to hashing it is possible to get an index that can tolerate some spelling variations for similarly sounding names. A more general approach to fuzzy searching would require a hash transformation that can tolerate mismatches of bit attribute strings. This would allow a certain neighborhood of a given vector to be referred to the same hash table location. A possible realization of this approach has been described in [2]. It is based on reversing the regular error correction scheme for a perfect code, such as the Golay code (23, 12, 7). With this code, the whole set of 23-bit vectors—the vertices of a 23 dimensional binary cube—is partitioned into

2^{12} spheres of radius 3. So, a transformation that maps 23-bit strings to 12-bit centers of these spheres is able to tolerate certain dissimilarities in bit positions of the 23-bit strings. Presenting the attributes by means of a 23-bit template yields an assortment of 12-bit indices, which provide fault-tolerance facilities for fuzzy matching and retrieval.

A hash transformation using the Golay code decoding procedure is applied to neighborhood spheres of radius 1 surrounding 23-bit binary vectors. This transformation yields 6 hash indices in 86.5% cases and one hash index in 13.5% of the cases. So, on average the redundancy in indexing an information item introduced by this technique is about 6·0.865 + 1·0.135 = 5.3.To directly find binary strings deviating from a given key within Hamming distance 2 it would be necessary just to replicate the contents of the dictionary by a factor of 5.3 with a corresponding increase in the number of accesses. This scheme can be illustrated by the following example. Suppose we have two 23-bit vectors represented by two integers: 1036 ($2^{10}+2^3+2^2$) and 1039 ($2^{10}+2^3+2^2+2^1+2^0$). These numbers differ in only the two last bit positions. Their six hash indices turn out to be: 1036→(0, 1054, 1164, 1293, 1644, 3084); 1039→(527, 1054, 1063, 1099, 1293, 3215). The input vectors (1036 and 1039) are stored at the hash table locations specified by these 12-bit keys. Therefore, in searching for the 23-bit vector 1036, we access the hash location 1054 and find also the vector 1039. Thus, by using a few direct accesses, each "fuzzy" search (with Hamming distance 2) returns a neighborhood around the desired item.

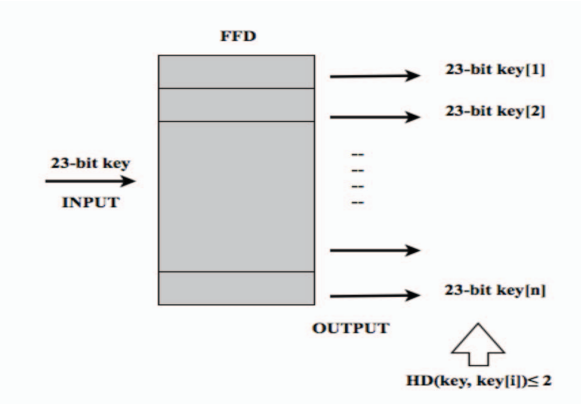

Fig. 3. General structure for the FuzzyFind Dicitionary

Implementation of the introduced data structure called FuzzyFind Dictionary has been described in [11]. This new data structure provides direct access to approximately matching information items, which are close in terms of Hamming's metric in binary attribute space. A dictionary is in essence a table that takes a key as an entry and if the key is present in the table returns some information associated with this key [11]. Therefore, FuzzyFind Dictionary would be considered as a table with an input of 23-bit key and outputs all the keys within Hamming distance 2 from the input key, as shown in Fig.3. The developed FuzzyFind Dictionary of about 100,000 words exhibits much better characteristics in comparison to implementation of this dictionary using other known methods. Typically, an average retrieval time per word is about 0.5 msec [11]. Therefore; as an enhancement, the suggested method (Pigeonhole search) would incorporate FuzzyFind Dictionary (FFD) in its indexing structure, in which it explained in more details in the next section.

IV. THE DEVELOPED SCHEME OF THE SUGGESTED METHOD WITH FUZZYFIND DICTIONARY

A. *FuzzyFind Dictionary organization*

Applying Golay code hash transformation as mentioned in the previous section will generate six hashes in 86.5% of the cases (lets call them Case A), and one hash in 13.5% of the cases (lets call them Case B). Also, we will replicate the contents of the dictionary by a factor of 5.3, as mentioned previously, in order to search for binary strings deviating from a given key within Hamming distance 2. In the first case (Case A), we will use pairwise concatenations of the indices, in order to get 15 addresses, (which is pair combinations of the six indices; C (6,2)), each address is of size 23 bits. One more address will be extracted from the pairwise concatenation of the " zero-index" with itself. This index is the first hash value of Golay code transformation. Thus, we will generate 16 addresses for Case A.

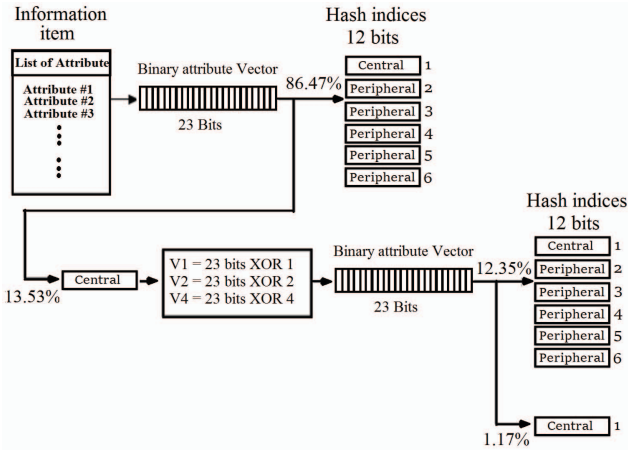

Fig. 4. The basic scheme for fault-tolerant hashing with Golay code transformation and FuzzyFind transformationm

Generally, a tolerance to 1-bit mismatch can be implemented by probing each hash index corresponding to all 1-bit modifications of a given word [2]. Thus, in Case B, we have only one 12-bit index, and this retrieval would require probing the hash values for the key modifications of the given key. These 1-bit modifications values are the numbers (1,2 and 4), and this what is shown in the Fig. 4, where we use bitwise

XOR to create a new vectors(V1, V2, V4). Then applying Golay code hash transformation on these vectors, will yield to two situations in Case B. In the first, we will got 6 indices and from those we will obtain 16 addresses (as in Case A), and this will occur in 12.35% as shown in Fig. 4. The second situation of Case B, is where we would have only one index, (just 1.17%), and from this we will obtain only one 23-bit address. Thus, we will have about 98.82% of the 23-bit vectors that have 16 address, and only 1.17% vectors with one address.

In certain circumstances, the 12-bit indices issued by two 23-bit vectors at Hamming distance 2 have two common values (as the example of (1036 and 1039)). Therefore, using pairwise concatenations of the indices it is possible to place information items with binary vectors at Hamming distance 2 in the same buckets. Thus 1036 and 1039 will have one common 23-bit address (as shown below), which is the result of the concatenation of the two common 12-bit indices.This would be implied to all vectors that have two distortions or less from each other.

1036 →	1039→
192	164320
480	164620
--	--
1969158	1969158
--	--
6540294	6306822

B. Implementiation of the developed Pigeonhole search.

Now we are going to Supplement FuzzyFind Dictionary into the suggested method (Pigeonhole search). Thus in this search, we will have a direct access to a dictionary of 23-bit words tolerating 2 mismatches. Suppose we have the same basic problem as in Fig.1; therefore, Applying Pigeonhole search, we will partition the Bit-Attribute Vector into segments of length 23 bits, as required by the FuzzyFind method. Then we will access each segment through its corresponding FuzzyFind Dictionary table. Therefore, we will locate the segments fast if they are within Hamming Distance 2 from given searching pattern. For each such match in each segment, we perform sequential lookup and compare the remaining segments within a specified threshold. Thus, according to the Pigeonhole Principle, if the specified Threshold = [(k*3)-1], (where $\mu = 2$), we will find all rows of Bit-Attribute Matrix within this distance. Fig. 6 shows a 92-bit attribute vector that partitioned into four segments, each of size 23 bits. Each segments is accessed through its corresponding FuzzyFind Dictionary table (in this case we have four FFD). Thus, all the matching locations of the suggested items will be gathered in a candidate list. This list will be searched sequentially and each item will be compared with the original 92-bit vector. Thus we will retrieve only the items that are within the specified threshold from the given vector. In the 92-bit vector example, the threshold equals to 11 mismatches.

This indicates that using FuzzyFind Dictionary (FFD) can have bigger tolerance of mismatches than the case of having at least one segment with exact match as in the 64-bit example (11 mismatches Vs. 3 mismatches). Another advantages is that the use of FuzzyFind Dictionary can have bigger size of segment and therefore, bigger size of Bit-Attribute Vector (92-bit Vs. 64-bit).

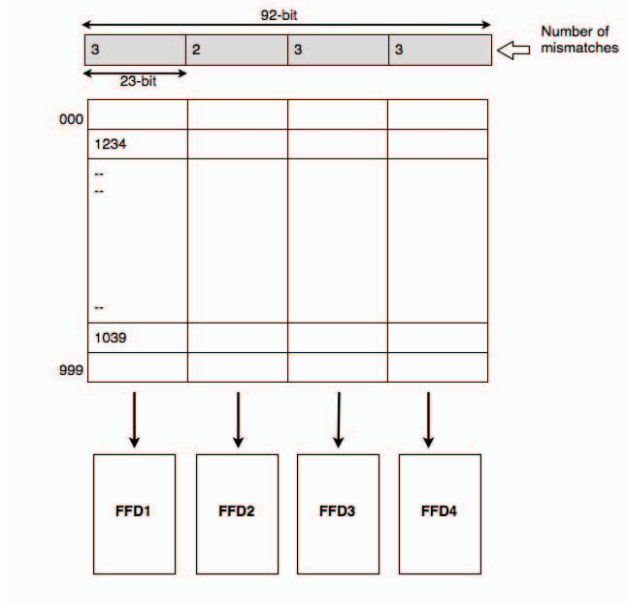

Fig. 5. The developed scheme of the Pigeonhole search with FFD

V. RESULTS

Typical experimental results for a simplistic case of 64-bit words tolerating 3 bit mismatches are shown below. Table (1) shows the time taken (in nanoseconds (ns)) by the sequential search versus Pigeonhole search. It also shows how much Pigeonhole search is faster than a corresponding sequential algorithm (speedup). The results show that Pigeonhole search is faster than sequential search by more than 6,000 times, when the file size is 10^7; and this what is demonstrated in Fig. 6 as well.

TABLE I. APPROXIMATE SEARCH RESULTS OF 64-BIT BASIC SCHEME

N	*Sequential Search*	*Pigeonhole search*	*Speedup*
10^3	130 ns	7 ns	19
10^4	970 ns	6 ns	160
10^5	9190 ns	6 ns	1,530
10^6	92571 ns	25 ns	3,700
10^7	892110 ns	140 ns	6,370

If we adjust the results of the 64-bit case to suits the new case of 92-bit vector tolerating 11 mismatches; and where we have FFD applied in the Pigeonhole search, we will have the following results in table (2). In the sequential search, searching for 92-bit vector, will be slower than serching for

64-bit vector by 1.5 times (92/64). While, in the case of Pigeonhole search, the developed scheme will be slower than the basic scheme by 16 times. Since in the basic sheme, accessing the table will take one unit of time. While in the devloped scheme with the FFD, the accessing will take 16 units of time, in order to access all the 16 addresses.

The results in Table (2) shows that the developed scheme of Pigeonhole search with FFD is hundreds times faster than regular sequential search. In contrast to initial example of 64-bit basic scheme, it allows searching with about 12% accuracy(11/92) rather than 4.5% accuracy(3/64).

TABLE II. APPROXIMATE SEARCH RESULTS OF 92-BIT DEVELOPED SCHEME WITH FFD

N	*Sequential Search*	*Pigeonhole search*	*Speedup*
10^3	195 ns	112 ns	2
10^4	1,455 ns	96 ns	15
10^5	13,785 ns	96 ns	144
10^6	138,857 ns	460 ns	347
10^7	1,338,165 ns	2240 ns	597

Fig.6 shows a comparison between the speedup for Pigeonhole search in both schemes (the basic 64-bit and the developed 92-bit scheme with FFD). Although the speedup for the new scheme is slower than the basic scheme, the new scheme is still faster than the regular sequential search by more than 500 times. It also, shows more accurate results and allows more mismatches tolerance, than the basic scheme.

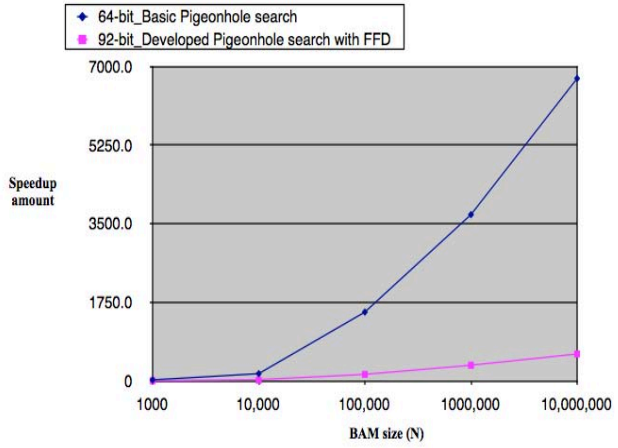

Fig. 6. Comparasion of Pigeonhole search speedup in both schemes.

VI. CONCLUSION REMARKS

This work contributes to the area of searching very large files with fuzzy criteria. Big Data systems would benefit from this technology and use it as a part of their sophisticated search operations, especially, in searching very large archives. Another interesting application where we can directly apply our technology is the QR code. QR code is a type of two-dimensional barcode that is in the form of the matrix code [14]. Given a full bit-template we have to find among a tremendous amounts of QR codes, a given one. This can be done with the developed scheme of our suggested method (Pigeonhole search), with about 13% accuracy. The other algorithms that have been used in QR code searching are not quite efficient; therefore, our suggested method will have an excellent contribution in such search. In general, Pigeonhole search is a practical technique that searches files beyond terabytes and still can perform much faster than traditional search algorithm like sequential search.

REFERENCES

[1] M. Crochemore and G. Tischler. The gapped suffix array: a new index structure for fast approximate matching. In String Processing and Information Retrieval, pages 359–364. Springer, 2010.

[2] S. Y. Berkovich and E. El-Qawasmeh. Reversing the error correction scheme for a fault-tolerant indexing. The Computer Journal, 43(1): 54–64, 2000.

[3] P. Clifford and R. Clifford. Simple deterministic wildcard matching. Information Processing Letters, 101(2):53–54, 2007.

[4] R. Cole and R. Hariharan. Tree pattern matching and subset matching in randomized o (nlog 3 m) time. In Proceedings of the twenty-ninth annual ACM symposium on Theory of computing, pages 66–75. ACM, 1997.

[5] S. Y. Berkovich and A. Hegazy. Matching string patterns in large textual files. In IEEE International Symposium on New Directions in Computing, pages 122–127, 1985

[6] M. Hadjieleftheriou and C. Li. Efficient approximate search on string collections. Proceedings of the VLDB Endowment, 2(2): 1660–1661, 2009.

[7] S. Berkovich, E. El-Qawasmeh, G. Lapir, M. Mack, and C. Zincke. Organization of near matching in bit attribute matrix applied to associative access methods in information retrieval. In APPLIED INFORMATICS-PROCEEDINGS-, pages 62–64, 1998.

[8] Berkovich, Simon, and Duoduo Liao. "On clusterization of big data streams." Proceedings of the 3rd International Conference on Computing for Geospatial Research and Applications. ACM, 2012

[9] M. Yammahi, C. Shen, S. Berkovich. Approximate Search in very large files using the Pigeonhole Principle, International Review on Computers and Software (IRECOS), Vol. 8, No. 12.

[10] E. Berkovich, Method of and system for searching a data dictionary with fault tolerant indexing, Jan. 23 2007. US Patent 7,168,025.

[11] S. Y. Berkovich, E. Berkovich, B. Beroukhim, and G. M. Lapir. Organization of automatic spelling correction: Towards the design of intelligent information retrieval systems. The 21st National Conference of the ASEM, (Washington DC, pages 525– 527, 2000).

[12] S. B. Maurer and A. Ralston. Discrete algorithmic mathematics. (Addison-Wesley Reading MA, 1991).

[13] Kukich, Karen, 1992. "Techniques for Automatically Correcting Words in Text," ACM Computing Surveys, Vol. 24, No. 4, pp. 377-43

[14] Poomvichid, T., Patirupanusara, P., & Ketcham, M. (2012, August). The QR code for audio watermarking using Genetic algorithm. In Proceedings of the International Conference on Machine Learning and Computer Science (IMLCS'12) (pp. 171-174).