Investigation of Golay Code (24, 12, 8) Structure in Improving Search Techniques

Eyas El-Qawasmeh[1], Maytham Safar[2], and Talal Kanan[3]
[1]College of Computer and Information Sciences, King Saud University, Saudi Arabia
[2]Department of Computer Science, Jordan University of Science and Technology, Jordan
[3]Department of Computer Engineering, Kuwait University, Kuwait

Abstract: *This paper presents a new technique for hash searching that is designed for approximate matching problem of multi-attribute objects. The suggested technique can be used for improving the search operation when the multi-attribute objects are partially distorted or when the searching criterion is not specified properly. The suggested approach is based upon reversing the conventional scheme of Golay code (24, 12, 8), which maps 24-bit vectors into 12-bit message words. In this technique, a multi-dimensional space is used to represent objects, where each object is given by a 24-bit vector. The closeness of the objects is determined by partitioning a 24-dimensional cube. In addition, the possibility of 1-bit distortions is considered through bit modifications of the 24-bit vectors. Thus, we proposed a hash table of 4096 entries that is fault-tolerant in the sense that the index is the same for any two different 24-bit vectors that share the same sphere. This allows organizing a direct retrieval of a neighborhood of 24-bit vectors with two or possibly more mismatches. The simulation experiments measured the retrieval capabilities of the proposed system and the expected hash distribution.*

Keywords: *Golay code, information retrieval, distortion, and hashing.*

Received December 25, 2008; accepted August 3, 2009

1. Introduction

Hashing is a technique that is used in information retrieval since it requires a linear time of complexity in most cases. One of the most important characteristics of the hash searching techniques is the hash function. Several hash functions are being used widely in order to achieve the required performance, storage reduction and simplicity. Most of the functions are simple mathematical ones; such as the modulus. In this paper we suggest a new function to be used in the Hash searching techniques. The function is based on the decode operation of the famous Golay code (24, 12, 8) error detection and correction technique, which also known as extended Golay code. We aim at studying the performance characteristics along with the search capabilities of the proposed design.

Hash searching techniques are widely used in information retrieval, especially when the searched key is known exactly as it is stored. However, there are situations where only partial information of the object is available. Therefore, approximate matching algorithms should be implemented.

Approximate matching applies to databases that store objects with several real-valued attributes. The approximate matching is also referred to as the nearest neighbor problem. The dissimilarity or distance measures may solve the problem of finding those objects in the databases that are most similar to a given query [10]. For example, the name "Mohammed" has different spelling variations when it is written in English such as "Mohammad", and "Mohamad". Approximate matching is defined as: Given a collection of N objects (each of which is described by k real-valued attributes) and a dissimilarity measure D, find the m objects closest to a query (possibly not in the collection set) with specified attribute values [2, 6].

Among the solutions that try to handle the approximate matching problem are Soundex Technique, and NYSIIS technique [9]. Both of them are designed for approximate matching in names only. Soundex uses the phonetic approach in order to locate all similar sounding names. In the Soundex method, many variations are completely merged, while some remain distinct [9]. NYSIIS uses the same coding concept in a different style.

This paper proposed a new technique for hash searching, in which we define a new hash function that is based on the decode operation of Golay code (24, 12, 8). The proposed technique suggests to use attribute vectors to represent any type of data, including but not limited to, personal records, vehicle information, and relational database systems. The attribute vectors are created by answering a set of Yes/No questions and setting/resetting the sequence of bits accordingly. The proposed technique also tolerates bit distortion that may occur in the attribute vectors during the search operation.

The organization of the paper will be as follows. Section 2 presents the proposed scheme. Section 3 presents the partitioning of a 24 binary cube. Section 4 is the Golay construction. Section 5 is the distribution analysis. Section 6 is the proposed design. Section 7 is the retrieval capabilities. Section 8 is the average number of probes. Section 9 is a discussion, and finally section 10 is the conclusion.

2. Proposed scheme

Our work suggests reversing the conventional application of error-correction codes. In error correction codes, a sequence of bits that we need to transfer is converted to a longer sequence of bits called codeword by adding some extra bits. This codeword is transmitted over a channel. The receiver site will be able to retrieve the original sequence even if the original sequence (message) has some bits of error (1-4) bits depending on the used error codes. We consider the decoding procedure as the primary operation, and we expect that a neighborhood of codewords may be mapped into a smaller collection of datawords. Hash indices, which enable searching for binary vectors, are to be constructed from these datawords. The techniques proposed in this paper will use Golay code for error correction and detection.

Golay code has two variations. The first one is Golay (23, 12, 7). In this code, if we want to send a 12-bit message, then we add additional 11-bits to it. The total will come to 23-bits. Now, we send it. The receiver side will be able to convert the 23-bits message to its original 12-bits message, even if there is a 3-bit distortion. This is applicable to all possibilities of the 23-bits and because of this; it is called prefect Golay code (23, 12, 7). However, there is another variation called Golay code (24, 12, 8). This code can detect up to 4-bits, but it can correct 3-bits distortion. If there is 4-bits distortion, then it might correct these 4-bits, but there is no guarantee that it will be able to fix it. This code is code not perfect code.A searching technique tolerating 1-bit mismatch can be implemented by brute-force, which can be achieved by probing each hash index that consist of all 1-bit distortions of a given word. Without using brute-force, dealing with 1-bit distortions of a certain 24-bit key will involve inspecting 8 hash buckets (one bucket is determined by the hash value of this key and the remaining 24 buckets are determined by the hash values of all 1-bit distortions of the word). The proposed scheme that employs error-correction codes for fault-tolerant retrieval is useful in such a way that it eliminates the performance degradation of the brute-force procedure.

Our contribution is that suggested method will benefit from the characteristics of a non-perfect code. A perfect code is one in which there are equal radius spheres surrounding the codewords; the spheres must also be disjoint and completely fill the space [1, 9]. In the Golay code (23, 12, 7), this is equivalent to the property that every word with a length of 12 bits has a distance of at most 7 from one and only one codeword. The Golay code (23, 12, 7) is perfect and the codewords represent all 2^{23} = 8388608 23-bit combinations. Golay code (24, 12, 8) is a non-perfect code since there exists a subset of codewords in the space where each codeword can be decoded to more than one dataword.

3. Partitioning of a 24 Binary Cube

Consider a 24-dimensional cube that contains exactly 2^{24} different points, where each point in this cube can be represented by a 24-bit binary vector. The task is to partition this 24-dimensional cube into spheres. A desirable partition will be a partition that satisfies the following characteristics:

1. All of the spheres are of equal radius.
2. All of the points in this 24-dimensional cube are included by a partition such that each point belongs to only one partitioning sphere.

The 24-dimensional cube cannot be partitioned in such a way where both of the properties can be satisfied. To be more specific the whole cube can be partitioned into 2^{12} = 4096 spheres, which are referred to as partition spheres, and they are of equal radius but they do not include all the points. Golay (24, 12, 8) is not a perfect code, but still we can benefit from its characteristics tolerating some bit distortions to achieve good results. The purpose here is to divide the cube into smaller spheres so that the search operation can be performed using only a subset rather than the whole cube meanwhile tolerating bit mismatches. Each partition sphere can be identified by its center, which is a 24-bit codeword that can be converted to a unique 12-bit dataword. Any two different partition spheres among the 2^{12} spheres will have two different 12-bit data messages associated with them. These centers are to be treated as the hash indices, the datawords that do not belong to certain spheres will have the opportunity to fit in one of multiple spheres with the same probability. The reason for this is due to the non-perfect property in Golay code (24, 12, 8). Converting from a 24-bit codeword into a 12-bit data message is called decoding, while converting from a 12-bit data message to the corresponding 24-bit codeword is called encoding.

The results of partitioning the 24-dimensional cube are 2^{12} different partitioned spheres. A 24-bit vector represents each point within the partitioned sphere and one of these points is the center of the sphere. Upon examination of any partitioned sphere, one will realize that all of the codewords in a given sphere are at a Hamming distance ≤ 4 from its center.

4. The Golay Code Construction

Extended Golay Code is also known as Golay code (24, 12, 8), where we have codewords of length 24 bits describing the original 12-bit message. The minimum

Hamming distance between any two codewords is 8. The 24 Golay code is an extension of the 23 Golay code. Golay code (24, 12, 8) guarantees retrieving the original data if the error occurred is three bits or less [4, 8]. If errors occurred in four bits there is no guarantee to recover the original data, however, it is possible, due to the fact that the decoding may result in having the original words relate to a group or another with, perhaps, the same probability. This will benefit the approximate matching and similarity measures to study a new approach to the matching problem of multi-attribute objects. In addition, we will study the possibility of near matching by changing one bit at a time of all the 24 bits and see the effect of this. This technique can be used to improve Information Retrieval when the multi-attribute objects are partially distorted or when the searching criterion is not specified properly.

Reversing the technique of Golay (24, 12, 8) means that we will pick up 24 bits and decode them to form a 12-bit data word. In the case of at most three erroneous bits existence, this will give us good results. The 24 bits can represent the attributes of objects, documents, records, and/or any piece of information. This leads to having 2^{24} = 16,777,216 numbers targeting only 2^{12} = 4096 numbers. In this paper, we tend to address searching the best match where the key is almost correct in information retrieval systems.

5. Distribution Analysis

The number of points in the 24-dimensional cube is 2^{24}. Any 24-bit codeword can be represented by one point in this cube, which has a value between 0 and 2^{24}-1 (0 to 16,777,215). The representation of every codeword contains a number of 1's that ranges from 0 to 24; this is called the weight of the codeword. A piece of script was written to manipulate and collect the data, which simply calculates the distinct number of words against the possible codeword weight. The result of running this script can be seen in Figure 1.

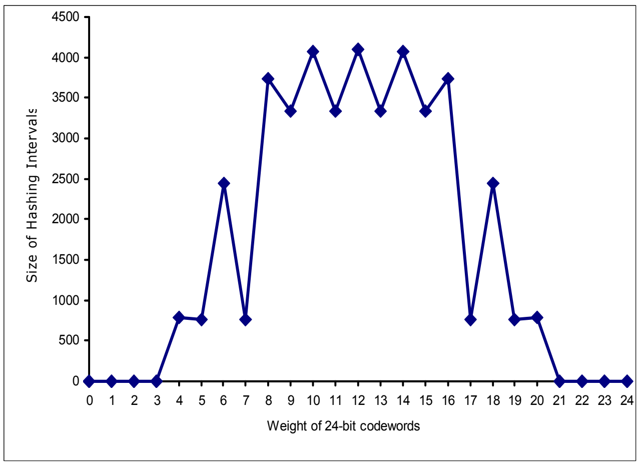


Figure 1. Number of distinct words per codeword weight.

The distribution shown in Figure 1 is not scattered uniformly. Evident from the distribution is that the weight determines the scattering pattern. For example, vectors with a weight of 5 spread over 759 different locations, where vectors with a weight of 11 spread over 3335 different locations. Figure 1 also shows that the number of hash indices is small (closed to zero, but not zero) at both ends of the distribution (see values for weights 0, 1, 2, 3, 21, 22, 23 and 24).

In fact all 24-bit vectors with weight 3 or less contribute to the hash index of 0, correspondingly, when the weight is 21 or more. In other words, when the number of 0's is less than 4, the decoded 24-bit vector will map to hash index 4095. This constitutes the worst case, which is considered to be the bucket size with a maximum number of entries in it.

The existence of 24-bit vectors with a minimal number of 1's (or correspondingly 0's) is rare in the cube. For instance, the number of 24-bit vectors in the 24-dimensional cube with a weight of 2 is exactly $\binom{24}{2} = 276$.

These points constitute much less than 1% of the 24-dimensional cube points and are indexed to only one partition sphere. In contrast, cases such as 24-bit vectors with a weight of 12 occur more commonly. In this case, the number of 24-bit vectors is exactly

$$\binom{24}{12} = 2704156 .$$

These vectors represent more than 16% of the whole space and are indexed to 4094 different locations in the hash table. After that, we modified the script to perform 1-bit distortion on the codewords before decoding. The experimental results showed that the number of hash indices increases after including the distortion.

6. Proposed Design

The proposed design generates a 24-bit vector for a multi-attribute object, which represents the characteristics of the object. We will use the object properties to create the vector; each property will be represented by one bit, or more. Also, we can represent two properties or more by one bit. When an object shows a specific property, the binary 1 will be placed in the corresponding bit; otherwise the default 0 will be placed in that bit. The set of property bits requires exactly 24 bits, but the order of the bits is not important. For an object requiring more than 24 bits, we suggest selecting the most distinguishable properties or the set of properties that can be broken into two sets, each one consisting of 24 bits.

We have implemented a simulation program for misspelled names using the proposed technique. The program, simply, creates a 24-bit vector for each name upon storing, decodes the vector and stores the name in all the decoded hash indices. Upon searching, a misspelled name is also associated to a 24-bit vector, the vector is decoded and all the hash indices are probed. Constructing the 24-bit vectors is achieved by answering a set of 24 (Yes/No) questions, each answer is assigned to a bit position to be either 0 or 1. The set of questions depend on the nature of the data to be stored and the properties of the items. Even, choosing the name properties might differ slightly from one culture to another depending on the nature of the language.

A system using the proposed technique consists of two basic operations; the store, and the search. The implementation should begin by defining the properties and their corresponding bits. The next step is transferring object attributes to the corresponding 24-bit vector. Another script was written to generate all distinct hash indices for the object and all of its 1-bit distortions, the number of distinct hash indices as for the whole space is shown in Table 1.

Table 1. Number of hash indices percentage.

No of Hash Indices	Percentage
1	7.35 %
6	12.08 %
7	31.15 %
11	5.37 %
12	19.43 %
13	24.71 %

Table 1 is generated by decoding all the possible numbers ranging from 0 to 2^{24} -1, along with each numbers' 1-bit distortions. The distinct number of datawords were collected and kept. In our context, this means that each codeword will be stored in 1, 6, 7, 11, 12, or 13 hash indices. On average 9.1 hash indices, calculated as follows: $1 * 0.0735 + 6 * 0.1208 + 7 * 0.3115 + 11 * 0.0537 + 12 * 0.1943 + 13 * 0.2471 = 9.1134 \cong 9.1$. From now on, we will use the numbers 9.1 or 9 to indicate the redundancy of average storing each and every codeword.

The hash indices are used to store the objects. The hash table will be organized using the chaining approach. In this approach, each hash index contains a pointer to a linked list that is allocated outside of the hash table. This linked list will be used to store all the object information that has been hashed to its corresponding index.

For the search operation the user needs to find all hash indices for the 24-bit search query and all possible 1-bit distortions. Upon determining the hash indices of the object, the corresponding linked list can be traversed through the hash index pointer [5]. Thus, the store operation has the following steps:

1. Find the 24-bit representation of the multi-attribute object.
2. Decode the 24-bit vector and store the object in the linked list associated with this index.
3. Consider all possible 1-bit distortions of the object and decode each 24-bit vector after the distortion.
4. Find all distinct hash indices generated from steps 2 and 3.
5. Store the object in the corresponding linked list of the indices.
6. Likewise, the search operation consists of the following steps:
7. Find the 24-bit vector of the search query.
8. Decode the 24-bit vector.
9. Find all possible 1-bit distortions.
10. Decode each possible distortion.
11. Find all distinct hash indices of the 25 decoded values (1 original and 24 with 1-bit distortion).
12. For all distinct hash indices, traverse the corresponding linked list.
13. Determine if an exact match exists and return it, otherwise return the whole traversed linked lists.

Table 1 shows that 31.15% of decoding processes, applied to the vectors and their 1-bit distortions, generated seven hash indices; Central 1, Peripheral 2, Peripheral 3, Peripheral 4, Peripheral 5, Peripheral 6, and Peripheral 7. The utilization of this searching scheme begins with filling the hash table. The hash table presents an array of $2^{12} = 4096$ pointers to the buckets with the 24-bit keys. Both kinds of binary vectors, the 12-bit indices to the array and the 24-bit keys, are represented as unsigned long integers by 32-bit words. For example, a 24-bit vector (0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,1,1,0,1) with 1 in positions 5, 3, 2, and 0 is represented as an unsigned long integer: $45 = 2^5 + 2^3 + 2^2 + 2^0$. By decoding this 24-bit key, we achieve 6 unsigned long integers as 12-bit hash indices; namely 2336: (1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0), 1176: (0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0), 514: (0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0), 64: (0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0), 5: (0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1), and 0: (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0). These 6 numbers are used as addresses for the 6 buckets where the object information related to key 45 is to be inserted.

7. Retrieval Capabilities

The basic searching scheme considers the 24-bit vector and all its possible 1-bit distortions during the creation and the search routines. The matching retrieval capabilities of the basic scheme were computed. Before we take a look at the results of the retrieval capabilities, we need to mention that the shared hash indices that were calculated represent the Recall. Recall is one of the measurement tools widely used in information retrieval and search techniques. It can be defined as the number of the retrieved correct matches to the total number of matches. Figure 2 plots the matching retrieval capabilities of the basic searching scheme. We ought to mention here, that we were not able to achieve a full run to the previous algorithm, due to the fact that this run having step 1 in the main loop takes many years to be completed. So, we managed to make several runs and generalize the results. The largest run we were able to achieve was using step 600. The counter was incremented, after each iteration, by a random number ranging from 450 to 750.

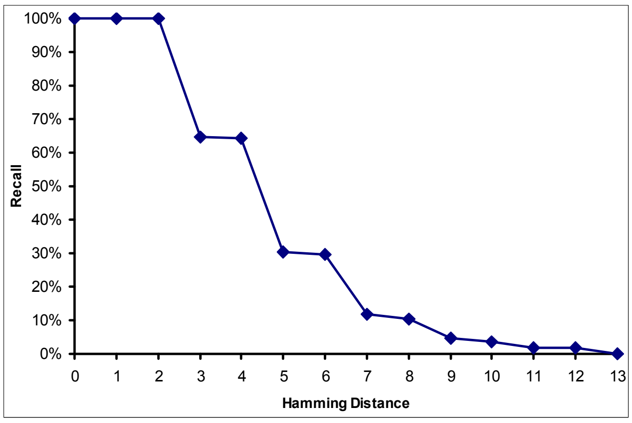


Figure 2. Matching retrieval capabilities of the basic searching scheme.

As clearly appears in Figure 2, the technique offers 100% recall at Hamming distances 1 and 2 and less recall at higher Hamming distances. However, we found that the cost for this is that we will have 8.02 replication in storage. Although the replication is a disadvantage, but the gain is that the search operation becomes faster. The gained speed is due to the fact that only 9.1 (on average) linked lists are searched instead of the whole set of objects.

8. Average Number of Probes

It should be noticed that the number of hash indices generated from decoding a 24-bit vector that has a deviation of less than 3 will always be decoded to one and only one hash index, while any 24-bit vector with a deviation of 3 will always be decoded to more than one hash index. This was verified by decoding all 24-bit vectors in the 24-dimensional cube.

We have implemented a simulation experiment to evaluate the searching time for the searching scheme, which provides an acceptable compromise between the redundancy and the retrieval characteristics. The object is stored in six, seven, eleven, twelve or thirteen different buckets using the pointers of the hash indices. A search operation utilizes these hash indices in order to traverse those buckets.

The experiment to find the number of probes starts by simulating the creation of a file of 4096 entries. The entries are built such that their average number of 1's equals an exact value ranging from 1 to 23. The search is performed using the same selected value. The results of this experiment are plotted in Figure 3. For example, selecting a value such as 4 means that the majority of the 24-bit vector entries have four attribute bits set to 1. The number of 1's in the remaining 24-bit vector entries will be close to four, such that many of them will have three or five 1's, and less will have two or six 1's and so forth. Larger numbers of entries can be chosen, but the normalization process, in terms of 4096, shows that the results are scaled and so, normalized large entries behave like a file of 4096 entries. As an example, let us select a weight like 7, we found that the number of probes is almost 210, which means that for a system with an average number of 1's equal to 7, the number of probes need to be accessed is equal to 210. The number 210 is a result of the summation of all probes. It is obvious that scanning 210 entries involves less efforts than scanning 4096/9.1 ≅ 450 entries.

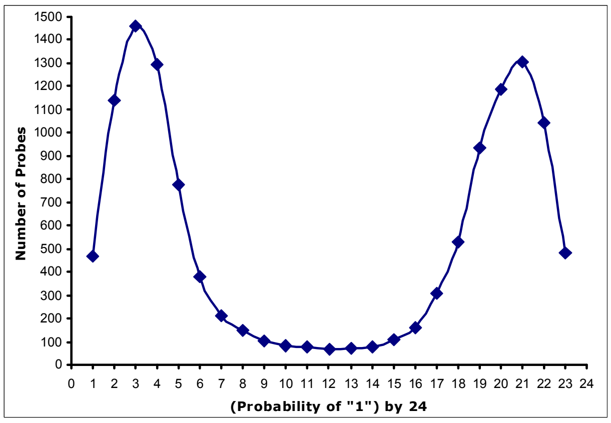


Figure 3. Number of probes for all hash indices without any restrictions.

The previous average search of the basic searching scheme may be enhanced with some adjustments. One method of improvement is to create two hash tables instead of one. The first table will be designed for the central hash index, regardless of whether there is one hash index or more than one hash index for the 24-bit vector and all its 1-bit distortions. The second table will use the remaining hash indices. The central index will be the hash of the 24-bit vector without any distortion. Each table will contain exactly 4096 indices and they will be treated in the same way as the single-table was treated. The creation of two tables reduces the average search time.

Another way to improve the average search is to unconditionally allow all 24-bit vectors, but to treat hash indices 0 and 4095 separately. These two indices, which increase rapidly for systems with a small number of 1's (correspondingly high number of 1's) can take advantage of the computed addresses of the table-of-errors.

An attempt to improve the average search can also be achieved by using mixed re-coding. This involves using opposite re-coding, where 1 indicates the absence of the attribute. This trick for re-coding is only recommended for the two attributes with the lowest probabilities. In summary, we can say that the suggested technique is suitable for systems with average weights ranging from 6 - 18. For systems outside this range, the user has several choices. The options include splitting the hash table, treating rare weight vectors specially, handling hash indices 0 and 4095 separately, and performing bitwise XOR on some specific bits. Combining more than one of these improvements will also enhance the performance.

9. Discussion

The search scheme provides an acceptable trade-off between the time-space redundancy and the retrieval characteristics. Each key is stored in 9.1 different buckets of the hash table on average and correspondingly, searching requires traversing these 9 buckets. The presented scheme expands the capabilities of ordinary hashing by introducing "fault-tolerance". On the positive side, speed is gained by using this technique. On the negative side, additional memory is required. Searching with this scheme for a given binary vector yields the whole neighborhood of this vector at a Hamming distance of 2. A certain portion of binary vectors at a greater distance would also be retrieved.

Evaluation of the performance of this searching scheme entails different choices for searching, depending upon various statistical characteristics of the binary attribute vectors. Additional concerns regarding the uniformity of hash transformation should be considered. The user should consider the influences of patterns which may shrink the scattering interval. The distribution analysis explained how vectors of a specific weight scatter over the hash table. Thus, this technique is mostly suitable for systems where the number of attributes is in the range from 5 to 19. If the number of attributes of objects in a system is predominately around 4, the performance of the search will degrade. When the predominant number of attributes is 3 or less, the performance of the system will degrade to a sequential look-up.

The performance of a hashing scheme is determined primarily by the average size of the buckets. For ordinary hashing, a set of 4096 random uniformly distributed keys would be accommodated in the hash table in buckets of a small average size. In using fault-tolerant hashing, the insertion of 4096 keys in the 4096 positions of the hash table results in 9 times the redundancy. Ideally, we would get 4096 buckets with a size of 9, or more accurately with a size 9.1. However, the average bucket size exceeds this redundancy coefficient, because hash values do not get a complete scattering over their available range. Thus, the shrinking of the scattering interval for low weight vectors (correspondingly high weight) increases the average bucket size. A preliminary performance evaluation was presented previously. The searching operation may be enhanced with some adjustments to the basic searching scheme. The first modification is performing the creation and the search for vectors only in the range of 4-20. This will improve the performance as a result of the experiments that we conducted. Vectors outside this range can be treated separately, since they are not common. This treatment can use the common approach and the non-common approach is treating the rare cases separately.

A simulation program was written to observe this behavior; it reads a name and computes the hashes of this name. Following this, another name (which will most likely be misspelled) will be read, and the calculated hash indices of it will be compared with the previous name. The program finds the common indices between the hashing of both names, as well as how many bucket access attempts will be required for retrieval.

10. Conclusions

This work contributes to establishing an approximate equivalence of information items. For this purpose, we have developed a technique of fault-tolerant hashing based on reversing the conventional usage of error-correction codes. In this technique, an error-correcting decoding procedure is applied to a certain neighborhood of binary attribute vectors. This application leads to the creation of a relatively smaller set of reference indices. The developed construction employed the Golay code (24, 12, 8). This code provides a non-perfect partitioning of a 24-dimensional binary cube and yields 13, 12, 11, 7, 6 indices or 1 index for a neighborhood of size 1 in Hamming's metrics. These indices are fault-tolerant in the sense that they allow the identification of information objects with certain mismatched attributes. The suggested approach can be generalized using error-correction codes other than the Golay code. This overcomes the 24-bit restriction on the size of the binary attribute vectors, but would result in a less regular structure of the reference indices.

In Information Retrieval, the suggested procedure is considered to be a special kind of hash coding transformation. With this transformation, information objects with mismatched attributes can be mapped to the same location and treated as "approximately equivalent". Hash coding is used primarily for searching, but it also has a variety of other applications. With unique capabilities for identification of approximately equivalent objects, the effectiveness of hash coding applications can be expanded.

Besides a direct application to information retrieval, the developed technique is beneficial for many complex computational procedures that incorporate Approximate Matching operations. Typical procedures of this kind include recovering close matches from vector-quantization tables and finding similarities in a population of binary strings for genetic algorithms.

Near-matching capabilities for binary vectors can be adapted to treat approximate matching of multi-dimensional objects with numerical components by encoding the values of these components in Golay code.

References

[1] Christos F., "Access Methods for Text," *Computer Journal of ACM Computing Surveys*, vol. 17, no. 1, pp. 49-74, 1985.

[2] Simon B. and Eyas Q., "Reversing the Error-Correction Scheme for Fault-Tolerant Hash Indexing," *Computer Journal*, England, vol. 43, no. 1, pp. 54-64, 2000.

[3] Florence M. and Neil S., *The Theory of Error-Correcting Codes*, Elsevier Science, 1977.

[4] Henrik L., "Error and Traffic Control for High-Speed Networks," *Dissertation*, Royal Institute of Technology (KTH), 2005.

[5] James R., "Equivalence Classes in the Real World," *Pi Mu Epsilon Journal*, vol. 9, no. 2, pp. 579-583, 1993.

[6] Jerome F., Jon B., and Raphael F., "An Algorithm for Finding Best Matches in Logarithmic Expected Time," *Computer Journal ACM Transactions on Mathematical Software*, vol. 3, no. 3, pp. 209-226, 1997.

[7] Joseph P. and Antonio Z., "Automatic Spelling Correction in Scientific and Scholarly Text," *Computer Journal Communications of the ACM*, vol. 27, no. 4, pp. 358-368, 1984.

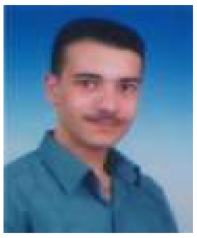
[8] Shu L. and Daniel C., *Error Control Coding: Fundamentals and Applications*, Prentice Hall, 1983.

[9] Tuvi E., Alexander V., and Er V., "Perfect Binary Codes: Constructions, Properties, and Enumeration," *Computer Journal IEEE Transactions on Information Theory*, vol. 40, no. 3, pp. 654-763, 1994.

[10] Vladimir B., "Hashing of Databases Based on Indirect Observations of Hamming Distance," *Computer Journal IEEE Transactions on Information Theory*, vol. 42, no. 2, pp. 664-671, 1996.

Eyas El-Qawasmeh received his BSc degree in Computer Science in 1985 from Yarmouk University, Jordan. He then joined the Yarmouk University as teaching assistant at the Computer Science Department. In 1992, he joined the George Washington University, Washington, DC, USA where he obtained his MS and PhD degrees in Software and systems in 1994 and 1997, respectively. In 2001, he joined George Washington University, USA as visiting researcher through a Fulbright Commission grant. In 2001, he won Hijjawi Research Prize for Computer Science. His areas of interest include multimedia databases, information retrieval, and object-oriented. He has authored/co-authored over 70 research publications in peer reviewed reputed journals, and conference proceedings. In addition, he was a keynote speaker in many International events. He is the program chair and proceedings chair for many international conferences. In addition, he is the guest editor and a member of the editorial board of many journals. El-Qawasmeh is currently an associate professor at King Saud University, Saudi Arabia.

Maytham Safar is currently an associate professor at the Computer Engineering Department at Kuwait University. He received his PhD degree in computer science from the University of Southern California in 2000. He has one book, three book chapters, and over sixty five conference/journal articles. Current research interests include social networks, sensor networks, location based services, image retrieval, and geographic information systems. He is a senior member of IEEE since 2008, and the first Kuwaiti to become an ACM senior member in 2009. He is also a member of IEEE Standards Association, IEEE Computer Society, IEEE Geoscience & Remote Sensing Society, IADIS, @WAS, and INSNA. He was granted over eleven research grants from research administration at Kuwait University, and Kuwait Foundation for the Advancement of Sciences (KFAS).



Talal Kanan achieved his BSc degree in computer science in 1999 from Yarmouk University, Jordan. He then joined several private companies specialized in software solutions; namely in the research & development departments. In 2004, he joined the Jordan University of Science and Technology, Jordan, where he obtained his MS degree in computer science in 2007. His areas of interest include information retrieval, hashing and indexing, relational databases and object-oriented programming. He is currently heading the Enterprise Services unit and the Research and Development unit in one of the leading private software houses in the Middle East.