FORM 2

THE PATENTS ACT, 1970

(39 of 1970)

&

THE PATENT RULES, 2003

COMPLETE SPECIFICATION

(See Section 10 and Rule 13)

Title of invention:

Method and System for Error Control Coding Using Expander-like Codes

Applicant

TATA Consultancy Services Limited

A company Incorporated in India under The Companies Act, 1956

Having address:

Nirmal Building, 9th Floor,`

Nariman Point, Mumbai 400021,

Maharashtra, India

The following specification particularly describes the invention and the manner in which it is to be performed.

FIELD OF THE INVENTION

The present invention relates to error control coding. Particularly the invention provides error correction codes derived from projective geometry based graphs. More particularly the invention provides method and system for generating error correction codes derived from higher dimensional projective geometry based graphs.

BACKGROUND OF THE INVENTION

Retention of information in form of data is one of the core functions of the modern computer, and is provided by storage. Hence storage is a fundamental component of all modern computing systems. Now-a-days data storage commonly refers to mass storage, more particularly optical storages such as CD-ROM and DVD-ROM or magnetic storage like hard disk drives etc. In architecture parlance, such components are generally called secondary storage.

Disc storage is a general category of secondary storage mechanisms, in which data are digitally recorded by various electronic, magnetic, optical, or mechanical methods on a surface layer deposited of one or more planar, round and rotating platters. A disk drive is a device implementing such a storage mechanism with fixed or removable media. Internal hard disc drives are examples of fixed media, while CD-ROMs are example of removable media. In modern times, it is the disc storage that is the most popular way of implementing data storage unlike the tape storages in past.

The two popular methods of recording digital data on discs are magnetic and optical recording. Magnetic recording refers to the storage of data on a magnetized medium. Magnetic storage uses different patterns of magnetization in a magnetizable material to store data, and is a form of non-volatile memory. Hard disk drives, commonly found in a computer's CPU, is one example of magnetic recording. On the other hand, Optical Recording is encoding of binary digital data in the form of pits (point which lacks reflection when read) and lands (point that reflects when read) on a special material surface. The encoding material (e.g., Aluminum) sits atop a flat surface of thicker substrate (usually polycarbonate), which makes up the bulk of the disc. CD-ROM and DVD-ROM are the most prominent examples of optical disc recordings, while Laser Disc, Magneto-optical disc, Universal Media Disc, Blue-ray Disc, HD Disc, Holographic and protein-coated discs are other lesser known examples of optical recordings. The encoding pattern for most magnetic or optical recordings follows a continuous, spiral path covering the entire disc surface and extending from the innermost track to the outermost track.

Although discs are more durable than earlier storage mechanisms such as tapes, they are susceptible to environmental and daily-use damage. Unlike the now-obsolete 3.5-inch floppy disk, most removable media such as optical discs do not have an integrated protective casing and are therefore susceptible to data transfer problems due to scratches, fingerprints, and other environmental problems such as dust speckles. These data transfer problems, while the data is being read, manifests itself in form of bit errors in the digital data stream. Even mechanical issues such as vibration due to occasional high rotational speeds of disc motors also produce undesirable noise, and hence bit errors occur in fixed as well as removable media. A long sequence of bit read errors while a track is being read (e.g. a scratch on a track) can be characterized as burst error, while bit read error arising out of tiny dust speckle masking limited number of pits and lands on a track leads to random error. The occurrence of such events obviously not being rare, recovery of data to maximum extent in presence of such errors is an essential subsystem within most computing systems, such as CPU and disc players.

To achieve the data recovery caused by above said errors there is a need for efficient error correction coding. However, the existing methods and systems are capable of correcting bit errors to some extent but they are not efficient enough to correct burst error. Some of the error correcting systems and methodologies which form the prior art are given below:

US6842872 to Yedida, et al. provides a method for evaluating and optimizing an error-correcting code to be transmitted through a noisy channel and to be decoded by an iterative message-passing decoder. Yedida, et al. teaches the representation of error-correcting code by a parity check matrix which is further modeled as a bipartite graph having variable nodes and check nodes. Yedida, et al. specifically teaches about the analysis of decoder to obtain a set of density evolution rules including operators and operands, which are then transformed to projective operators and projected operands to generate a set of projective message passing rules. The projective message passing rules are applied iteratively to the error-correcting code modeled by the bipartite graph until a termination condition is reached.

The problem addressed particularly relates to evaluation and optimization of an error-correcting code modeled by the bipartite graph. Further it is concerned with soft decoding of LDPC codes. LDPC codes are bit error correcting codes which use message passing algorithm during each decoding step. The patent discusses about improving the reliability, estimated through projective analysis during each step of decoding. It doesn't teach about the implementation of the error-correcting code for improving burst error correction capabilities of decoders in conventional CD-ROM drives.

US5838695 to Yang provides a method and apparatus for processing a Reed-Solomon Product-like Code (RSPC) for error correction in a substantially real time mode. Yang specifically teaches about providing two RSPC error correctors in the CD-ROM decoder which performs RSPC error correction.

The problem addressed particularly relates to processing a Reed-Solomon Product-like Code (RSPC) for error correction applicable to CD-ROM format. Hence it is concerned with RS coding scheme for CD-ROM applications. The other focus of the scheme is to improve the speed of decoding process. It doesn't teach about providing error correction codes derived from projective geometry based graphs for improving burst error correction capabilities of decoders in conventional CD-ROM drives.

US5732093 to Huang provides a correction method and apparatus for optical disc systems, capable of performing error correction procedures. Huang specifically teaches a method of decoding the old data sequences and the associated erasure pointers so as to generate new data sequences and a set of erasure pointer modification parameters.

The problem addressed particularly relates to providing a correction method and apparatus for optical disc systems, capable of performing error correction procedures, and doesn't teach about providing error correction codes derived from projective geometry based graphs for improving burst error correction capabilities of decoders in conventional CD-ROM drives.

EP1111799 to Alexander, et al. provides an error-correction method for use in a process of decoding cross interleaved Reed-Solomon code (CIRC) that corrects errors in data stored as C1-code words $C1\_CDW_k$ (k=0, ..., 108) and C2-code words $C2\_CDW_m$ (m=0, ..., 108) in a memory with several locations N_{ij} (i=0, ..., 217; j=0, ..., 31), each of said locations N_{ij} containing a data byte of said data.

The problem addressed particularly relates to an error-correction method for use in a process of decoding cross interleaved Reed-Solomon code (CIRC) and doesn't teach about providing error correction codes derived from projective geometry based graphs for improving burst error correction capabilities of decoders in conventional CD-ROM drives.

FR2838581 to Emmanuel, et al. provides a method for decoding error-correction codes which are encoded as follows: A block of un-encoded data is encoded according to a global code comprising at least two constituent sub-codes (Ri). Emmanuel, et al. specifically teaches an

irregular bipartite graph is associated with the global code, the decoding method is iterative, and each iteration produces a block of extrinsic data.

The problem addressed particularly relates to method for decoding a block of coded data, coded according to a global code comprising at least two constituent sub-codes (Ri). It doesn't teach about providing error correction codes derived from projective geometry based graphs for improving burst error correction capabilities of decoders in conventional CD-ROM drives.

Yamauchi, et al. in "A 24×-speed CIRC decoder for a CD-DSP/CD-ROM decoder LSI" teaches a data recovery method for disk storage devices by applying matrix ECC (Error Correction Code) for a block of sectors. This method allows long burst data error correction capability with little loss of the data capacity and the performance. The configuration of a block is: j (integer) sectors are appended as the matrix ECC for leading k (integer) data sectors such as, Sector_1, Sector_2,,,, Sector_k, ECC_sector_1, ECC_sector_2,,,ECC_sector_j. Though with the teaching of Hiroyuki, et al. a data recovery method for disk storage devices is achieved by applying matrix ECC (Error Correction Code) for a block of sectors, however, the problem to provide error correction codes derived from projective geometry based graphs for improving burst error correction capabilities of decoders in conventional CD-ROM drives.

The above mentioned prior arts fail to disclose an efficient method and system for providing symbol error correcting codes. The prior arts discussed above also fail to handle error correction in large data blocks, particularly burst errors; instead most of them are bit error correcting method or systems. Thus, in the light of the above mentioned background art, it is evident that, there is a need for a solution that can provide error correction codes capable of correcting errors in large data blocks, particularly burst errors. The existing solutions generally do not provide support for correcting burst errors for disc-based secondary digital storage devices such as HDD, CD-ROM and DVD-ROMs. Hence, due to the drawbacks of the conventional approaches there remains a need for a new solution that can provide burst error correcting capabilities for disc-based secondary digital storage devices such as HDD, CD-ROM and DVD-ROMs.

OBJECTIVES OF THE INVENTION

In accordance with the present invention, the primary objective is to provide symbol error correcting codes based on a higher dimensional projective geometry graph.

Another objective of the invention is to provide a method and system for error control coding based on higher dimensional projective geometry graph that has exceptional average case performance for random errors when compared to the theoretical bounds of the code.

Another objective of the present invention is to provide symbol error correcting codes, which belongs to the family of expander-like codes having Reed Solomon (RS) codes as component codes.

Another objective of the present invention is to provide a method and system for error control coding which has exceptional random and burst error detection and correction capabilities for large data blocks in storage devices and in communication.

Another objective of the present invention is to provide a method and system for error control coding which is capable of correcting symbol errors with high degree of probability.

Yet another objective of the invention is to provide a method and system for error control coding that improves existing burst error correction capabilities of decoders within conventional CD-ROMs.

Still another objective of the present invention is to provide a method and system for error control coding which has a highly parallel and symmetric design.

SUMMARY OF THE INVENTION

Before the present methods, systems, and hardware enablement are described, it is to be understood that this invention is not limited to the particular systems, and methodologies described, as there can be multiple possible embodiments of the present invention which are not expressly illustrated in the present disclosure. It is also to be understood that the terminology used in the description is for the purpose of describing the particular versions or embodiments only, and is not intended to limit the scope of the present invention which will be limited only by the appended claims.

The present invention provides symbol error correcting codes based on higher dimensional projective geometry graphs.

In one embodiment of the invention a method and system is provided for error control coding based on higher dimensional projective geometry graph that has exceptional average case performance for random errors when compared to the theoretical bounds of the code.

In another embodiment of the invention the method and system is provided for error control coding based on higher dimensional projective geometry graph known as a bipartite graph. Typically, the said bipartite graph is derived from the incidence relations of a projective space.

In another embodiment of the invention, the said projective space of dimension *d* consists of one dimensional subspaces of a *(d+1)*-dimensional vector space. An *m*-dimensional subspace of the projective space consists of all one dimensional subspaces of a *(m+1)*-dimensional subspace of the vector space.

In another embodiment of the invention the symbols of the code are mapped to the edges of the bipartite graph in order to maximize the burst error detection and correction capacity. Each vertex of the graph has a fixed degree and the number of vertices in each partition of the graph is equal.

In yet another embodiment of the invention one vertex of the graph is associated with each *m*-dimensional subspace and one with each *(d-m-1)*-dimensional subspace of the said projective space. Two vertices are connected by an edge if the corresponding subspaces are incident on each other. Said vertices correspond to processors which comprise, means for decoding the edges incident on the vertex; ability to skip decoding if more than correctable errors are detected; and a memory element to store the address space in the required order of computation;

In still another embodiment of the invention the system for error control coding is having Reed-Solomon (RS) decoders with the capability of handling shortened RS codes.

The above said method and system are preferably error correcting codes for storage devices and communication but also can be used for many other applications.

BRIEF DESCRIPTION OF THE DRAWINGS

The foregoing summary, as well as the following detailed description of preferred embodiments, are better understood when read in conjunction with the appended drawings. For the purpose of illustrating the invention shown in the drawings are exemplary constructions of the invention; however, the invention is not limited to the specific methods and system disclosed in the drawings:

Figure 1 shows conventional/prior-art flow diagram of the process for error correction
Figure 2 is a flow diagram of method for error control coding using expander-like codes constructed from higher dimensional projective geometry based graphs
Figure 3 represents inter-dependence of variables and constraints in expander-like codes
Figure 4 illustrates ECC Block layout of DVD-R
Figure 5 represents ECC block's partition into recording frames

DETAILED DESCRIPTION OF THE INVENTION

Some embodiments of this invention, illustrating all its features, will now be discussed in detail.

The words "comprising," "having," "containing," and "including," and other forms thereof, are intended to be equivalent in meaning and be open ended in that an item or items following any one of these words is not meant to be an exhaustive listing of such item or items, or meant to be limited to only the listed item or items.

It must also be noted that as used herein and in the appended claims, the singular forms "a," "an," and "the" include plural references unless the context clearly dictates otherwise. Although any systems and methods similar or equivalent to those described herein can be used in the practice or testing of embodiments of the present invention, the preferred, systems and methods are now described.

The disclosed embodiments are merely exemplary of the invention, which may be embodied in various forms.

The present invention enables a method and system for error control coding based on a higher dimensional projective geometry graph and has exceptional average case performance for random errors when compared to the theoretical bounds of the code.

The present invention provides a method for error control coding for a digital storage device or packet data transmission. The said method is characterized by use of a symbol error correcting code based on at least one higher dimensional projective geometry based graph followed by mapping the symbols of the error correcting code to the edges of the said one or more graph, wherein the said higher dimensional projective geometry based graph is generated by the computer implemented steps of:

a. defining the projective space wherein the said projective space of dimension *d* consists of one dimensional subspaces of a *(d+1)*-dimensional vector space. An *m*-dimensional subspace of the projective space consists of all one dimensional subspaces of a *(m+1)*-dimensional subspace of the vector space;

b. deriving the higher dimensional projective geometry based graph from the incidence relations of said projective space, wherein the higher dimensional projective geometry based graph is having at least two partitions and each partition is having at least one vertex;

c. associating one vertex of the graph with each *m*-dimensional subspace and one with each *(d-m-1)*-dimensional subspace of the said projective space wherein each vertex of the graph has a fixed degree and the number of vertices in each partition of the graph are equal;

d. connecting all vertices from one partition with the other vertices from the other partition by an edge if the corresponding subspaces are incident on each other; and

e. decoding all symbols by all vertices in at least one partition of the higher dimensional projective geometry based graph, wherein the symbols correspond to the edges incident on the vertex; and coding for error control in a communication and digital storage device by use of a symbol error correcting code based on at least one derived higher dimensional projective geometry graph.

The present invention provides a system for error control coding in a digital storage device or packet data transmission; the said system comprising of at least one decoding device, at least one digital storage device and at least one memory element communicatively coupled with each other, wherein the said system is characterized by using a symbol error correcting code based on at least one higher dimensional projective geometry graph followed by mapping the symbols of the error correcting code to the edges of the said one or more graph, wherein the said higher dimensional projective geometry based graph is extracted from:

a. decoding device defining the projective space wherein the said projective space of dimension *d* consists of one dimensional subspaces of a *(d+1)*-dimensional vector space and an *m*-dimensional subspace of the projective space consists of all one dimensional subspaces of a *(m+1)*-dimensional subspace of the vector space;

b. decoding device deriving the higher dimensional projective geometry based graph from the incidence relations of said projective space, wherein the higher dimensional projective geometry based graph is having at least two partitions and each partition is having at least one vertex;

c. decoding device associating one vertex of the graph with each *m*-dimensional subspace and one with each *(d-m-1)*-dimensional subspace of the said projective space wherein each vertex of the graph has a fixed degree and the number of vertices in each partition of the graph are equal;

d. decoding device connecting all vertices from one partition with the other vertices from the other partition by an edge if the corresponding subspaces are incident on each other;

e. decoding device decoding all symbols by all vertices in at least one partition of the higher dimensional projective geometry based graph, wherein the symbol correspond to the edges incident on the vertex; and coding for error control in a communication and digital storage device by use of a symbol error correcting code based on at least one higher dimensional projective geometry graph.

Referring to **Figure 1**, it is a conventional/prior-art flow diagram of the process for error correction. The process starts at step 102; data is encoded and further recorded on digital storage such as CD-ROM, DVD etc. At step 104, data is decoded while reading digital storage. At step 106, errors are detected while decoding data from digital storage. At step 108, error correction codes are employed to correct errors while decoding data from digital storage. The process ends at step 110, errors that occurred while decoding data from digital storage are corrected.

Referring to **Figure 2,** it illustrates a flow diagram of a method for error control coding using expander-like codes constructed from higher dimensional projective geometry based graphs. The method starts at step 202, projective space is defined and choice of two subspaces is done, wherein one of the projective subspaces is of dimension *d,* consisting of one dimensional subspaces of a *(d+1)*-dimensional vector space and the other is a *m*-dimensional subspace of the projective space consisting of all one dimensional subspaces of a *(m+1)*-dimensional subspace of the vector space. At step 204, the higher dimensional projective geometry based graph is derived from the incidence relations of said projective space. The higher dimensional projective geometry based graph is having two partitions and each partition is having equal number of vertices. At

step 206, one vertex of the graph is associated with each *m*-dimensional subspace and one with each *(d-m-1)*-dimensional subspace of the projective space. Each vertex of the graph has a fixed degree and the number of vertices in each partition of the graph is equal. At step 208, one vertex from one partition is connected with the other vertex from the other partition by an edge if the corresponding subspaces are incident on each other. The method ends at step 210; one RS subcode is decoded by each vertex in at least one partition of the higher dimensional projective geometry based graph. The RS subcode corresponds to the edges incident on the vertex. The coding for error correction is done in digital storage device by use of a symbol error correcting code based on the derived higher dimensional projective geometry graph.

Expander codes

Expander codes are a family of asymptotically good, linear error-correcting codes. They can be decoded in sub-linear time (proportional to log (n), where n is length of codeword) using parallel decoding algorithms. Further, this can be achieved using identical component processors, whose count is linearly proportional to n. These codes are based on a class of graphs known as expander graphs.

Expander graph

An expander graph is a graph in which every set of vertices has an unusually large number of neighbors. More formally,

Let *G* = *(V,E)* be a graph with *n* vertices. Then the graph *G* is a δ-expander, if every set of at most *m* vertices expands by a factor of δ. That is,

$$\forall\, S \subset V : |S| \leq m \Rightarrow |\{y : \exists\, x \in S \text{ such that } (x,y) \in E| > \delta \cdot |S|$$

Expander codes being a subclass of LDPC codes, for whose iterative decoding using variables and constraints a bipartite graph is required, we are interested mainly in bipartite expander graph. More specifically, a general (unbalanced) bipartite expander graph is a *(c, d,Є, δ)* expander if it is a *(c,d)*-regular bipartite graph in which every subset of at most an Є fraction
of the *c*-regular vertices expand by a factor of at least δ.

The degree of “goodness” of expansion, especially for regular graphs, can also be measured using its eigenvalues. The largest eigenvalue of a k-regular graph is ‘k’. If the second largest eigenvalue is much smaller than ‘k’, then the graph is known to be a good expander.

Construction of Expander Codes

It is well known that a randomly chosen *(c,d)*-regular graph will be a good expander with high probability. A deterministic construction of good expander graph, that further leads to construction

of good expander codes is by considering the edge-vertex incidence graph B of a *d*-regular graph G. The edge-vertex incidence graph of G = (V,E), a *(2,d)*-regular bipartite graph, has vertex set E Ụ V and edge set

$$\{(e,v) \in E \times V : v \text{ is an endpoint of } e\}$$

Referring to **Figure 3,** vertices of B corresponding to edges E of G are then associated to variables (302), while vertices of B corresponding to vertices of G are associated to constraints (304) on these variables. Each constraint corresponds to a set of linear restrictions on the *d* variables that are its neighbors.

In particular, a constraint will require that the variables it restricts form a codeword in some linear code of length *d*. Further, all the constraints are required to impose isomorphic codes on different variables. The default construction of expander codes requires *d* to remain constant as the order of G increases. In invented construction, *d* will also increase with increase in order of G (hence the term, 'expander-like', instead of expander). However, all important properties and advantages of using specific expander codes for various applications still remain intact.

Formally, let B be a (2,d)-regular graph between set of n nodes called variables, and 2/d* n nodes called constraints. Let b(i,j) be a function such that for each constraint C_i, the variables neighboring C_i are $v_{b(i,1)}, \cdots, v_{b(i,d)}$. Let S be an error-correcting code of block length d. The expander code C(B, S) is the code of block length n whose codewords are the words (x1, · · · , xn) such that, for $1 \leq i \leq 2/d^* n$, $x_{b(i,1)}, \cdots, x_{b(i,d)}$ is a codeword of S.

Good Expander Codes

As pointed out earlier, the decoding algorithm for such codes is iterative. Hence good expander codes imply at least the following properties:

- Better minimum distance (hence larger error-correction capability) than other codes of same length,
- Faster convergence, and
- Better code rate than other codes in the same class

Construction of good codes having the above said properties is described below, Construction makes use of three theorems. We only state these theorems before giving the construction details. For proofs of the theorems refer literature.

We assume that an expander code C(B, S) has been constructed having S as a linear code of rate r, block length d, and **minimum relative distance** Є, while B as the edge-vertex incidence graph of a d-regular graph G with second-largest eigenvalue λ.

Theorem 1 The code C(B, S) constructed as above has rate at least *2r - 1*, and minimum relative distance at least

$$\left(\frac{\epsilon - \frac{\lambda}{d}}{1 - \frac{\lambda}{d}}\right)^2$$

Theorem 2 If a parallel decoding round for C(B, S) is given as input, a word of relative distance α from a codeword, then it will output a word of relative distance at most equal to

$$\alpha * \left(\frac{2}{3} + \frac{16\alpha}{\epsilon^2} + \frac{4\lambda}{\epsilon d}\right)$$

from that codeword.

Theorem 3 For all Є such that 1- 2H(Є) > 0, where H(·) is the binary entropy function, there exists a polynomial-time constructible family of expander codes of rate 1 - 2H(Є) and minimum relative distance arbitrarily close to $Є^2$ in which any α < $Є^2$ /48 fraction of error can be corrected by a circuit of size O(n log n) and depth O(log n).

From theorem 1, it is observed that to have high minimum relative distance for the expander code, S should have high Є and B should have low, λ/d.. Since B has been constructed out of d-regular graph G, low λ/d signifies high distance between first and second eigenvalues, i.e. the graph G has to be a “good” expander graph. Further, to have high rate for the expander code, S has to have a high rate r as well, besides having the requirement of having high minimum relative distance Є.

From theorem 2, it is observed that to shrink the relative distance of input word from the codeword after one iteration maximally, we need to again have Є as high as possible and λ/d as low as possible. Such maximal shrinking of distance, per iteration, leads to the fastest convergence possible, and is also brought out in the proof of theorem 3 which has been given in previous literature.

From theorem 3, it is observed that to be able to correct as high fraction of errors as possible, it is required to have Є as high as possible..

RS Codes as Good Component Codes

By choosing a "good expander" graph, and fixing a code with high minimum relative distance Є, one can design code, having the first two properties described earlier. Simultaneously, to have high code rate for C(B, S), the component code S also needs to have high rate r. Reed-Solomon codes are a class of non-binary, linear codes, which for a given rate, have the best minimum relative distance (so-called maximum distance separable codes), **and** vice-versa. The code parameters for the RS codes are given by (n, k, n − k + 1).

It had been observed that earlier definition of expander code requires 'd' to remain constant as 'n' increases. In the present construction, 'd' increases as 'n' increases. However, it is clear from statement of theorems 1 and 2 that higher value of 'd' leads to better properties of the code C(B, S). However, such codes may not be called expander codes in wake of definition of these, but just graph-based, or expander-like codes.

PG Graphs, Ramanujan Graphs and Good Expander Graphs

The construction of expander codes makes use of an unbalanced bipartite graph B made out of a d-regular graph G. Zemor pointed out that if G is a regular bipartite graph, then the % of errors that can be corrected using a parallel iterative decoding algorithm can be increased twelve-fold. Further, he reasoned out through theorem 1 that to achieve good minimum relative distance for the expander code graph G should be a Ramanujan graph with the property: λ (second-largest eigenvalue) ≤ $2\sqrt{d-1}$.

However, it should be noted that (a) Approximately half the constructions of the bigger class of Ramanujan graphs lead to bipartite regular graphs, and (b) Using bipartite regular graphs as G leads to twelve-fold improvement in error correction capability. Hence it is imperative that one focuses on using Ramanujan graphs for construction of good expander codes instead.

Construction of PG-graph based Expander-like Codes

In one embodiment of the invention to construct an expander-like code, Zemor's Construction has been followed. The present invention uses Zemor's construction, which is based on a d-regular balanced bipartite graph, G=(V,E). The set V is divided into two sets A and B, with |A| = |B| = n such that every edge has one endpoint in A and another in B. For any vertex t, the set of edges incident on t is denoted by E_t. As the graph is bipartite, the sets $E_t \ \forall t \in A E_t \ \forall t \in A$ induce a partition on E. A similar partition can be created using the edge sets of the vertices belonging to B. The expander code, C(G, S) is constructed by treating the edges of G as variables and the vertices as constraints for a binary component code S. The block length of code C is N = n * d. As

before, the second largest eigenvalue of G is denoted by λ. The steps in one decoding round of the algorithm suggested by Zemor are as follows:

- Each constraint t in set A completely decodes the sub-vector associated with the set of d variables, E_t, and replaces it with the closest codeword in S. This step can be carried out in parallel by all constraints in A as no symbol is shared between two constraints.

- The constraints in set B replace the sub-vector associated with its edge sets, E_t, $t \in B$t , with the closest codeword in S. This again can be carried out in parallel by all constraints.

In another embodiment of the invention we use the following two properties of projective space of dimension d over GF(s), namely P(d,GF(s)) where $s = p^k$, k being a positive integer for improving the error correction properties of the the said code beyond Zemor bound.

1. The number of subspaces of dimension m is equal to the number of subspaces of dimension d − m − 1.
2. The number of m-dimensional subspaces incident on each d −m− 1-dimensional subspace is equal to the number of d − m − 1-dimensional subspaces incident on each m-dimensional subspace.

In another embodiment of the invention the above said two properties of projective subspaces have been used to create balanced regular bipartite graphs under the condition q = 2, m=0,and d>2 (point-hyperplane incidence graphs). Point-hyperplane incidence graphs also satisfy the eigenvalue properties that make it a Ramanujan graph.

In another embodiment of the invention the following properties of Projective space over finite fields are used to count the number of hyperplanes and number of points on each hyperplane in a general projective space over finite fields. The description is also extended to count the number of points in an m-dimensional subspace and also to count the number of l-dimensional subspaces in an m-dimensional subspace where l<m

Though we restrict our construction of expander-like codes to $F = GF(2^k)$, the properties are detailed for general finite fields.

Consider a finite field F = GF(s) with s elements, where s is a power of a prime number p i.e. $s = p^k$, k being a positive integer. A projective space of dimension d is denoted by P(d, F) and consists of one-dimensional subspaces of the (d+1)-dimensional vector space over F (an

extension field over F), denoted by F^{d+1}. Elements of this vector space are of the form (x0, . . . , xd), where each xi ϵF. The total number of such elements are $s^{(d+1)} = p^{k(d+1)}$. An equivalence relation between these elements is defined as follows. Two non-zero elements x, y are equivalent if there exists an element λϵ GF(s) such that x = λ y. Clearly, each equivalence class consists of s elements of the field (s − 1 non-zero elements and 0), and forms a one-dimensional subspace. Such 1-d vector subspace corresponds to a point in the projective space. Points are the zero-dimensional subspaces of the projective space. Therefore, the total number of points in P(d, F) are

$$P(d) = \frac{\text{number of non-zero elements in the field}}{\text{number of non-zero elements in one equivalence class}} = \frac{s^{d+1}-1}{s-1}$$

An m-dimensional subspace of P(d, F) consists of all the one-dimensional subspaces of an (m + 1)-dimensional subspace of the vector space. The basis of this vector subspace will have (m + 1) linearly independent elements, say b0, …, bm. Every element of this subspace can be represented as a linear combination of these basis vectors.

$$\mathbf{x} = \sum_{i=0}^{m} \alpha_i b_i, \text{ where } \alpha_i \in \mathbb{F}(s)$$

Clearly, the number of elements in the vector subspace are $s^{(m+1)}$. The number of points in the m-dimensional projective subspace is given by P(m) defined in earlier equation. This (m + 1)-dimensional vector subspace and the corresponding projective subspace are said to have a co-dimension of r = d − m (the rank of the null space of this vector subspace).

Let us denote the collection of all the l-dimensional projective subspaces by Ω_l. Now, Ω_0 represents the set of all the points of the projective space, Ω_1 is the set of all lines, Ω_2 is the set of all planes and so on. To count the number of elements in each of these sets, we define the function

$$\phi(n,l,s) = \frac{(s^{n+1}-1)(s^n-1)\dots(s^{n-l+1}-1)}{(s-1)(s^2-1)\dots(s^{l+1}-1)}$$

Now, the number of m-dimensional subspaces of P(d, F) is Ø(d,m, s). For example, the number of points in P(d, F) is Ø(d, 0, s). Also, the number of l-dimensional subspaces contained in an m-

dimensional subspace (where $0 \leq l < m \leq d$) is Ø (m, l, s), while the number of m-dimensional subspaces containing a particular l-dimensional subspace is Ø(d − l − 1,m − l − 1, s).

In another embodiment of the invention one vertex of the graph is associated with each m-dimensional subspace and another one with each d − m − 1-dimensional subspace.

In another embodiment of the invention two vertices are connected by an edge if the corresponding subspaces are incident on each other. As edges lie only between subspaces of different dimensions, the graph is bipartite with vertices associated with m-dimensional subspaces forming one set and vertices associated with d − m − 1-dimensional subspaces forming another. Also, the two properties, listed above, ensure that both the vertex sets have the same number of elements and that each vertex has the same degree.

In another embodiment of the invention we consider the graph, G=(V,E) obtained by taking the points and hyperplanes of P(5,GF(2)). This is the first code constructed and used throughout. This projective space is generated from $GF(2^6)$. In this projective space, the number of points (= number of hyperplanes) is Ø(5, 0, 2)=63. Each point is incident on Ø(4, 3, 2)=31 hyperplanes and each hyperplane has Ø(4, 0, 2)=31 points. Therefore, we have |V |=126 and |E|=1953. This implies that the block length of code C is 1953 and the number of constraints in the code is 126. The second eigenvalue of G, λ is 4. Hence the ratio λ/d is quite small, as required for design of "good" expander-like codes.

In another embodiment of the invention as the expander graph G is 31-regular, the block length of the component code also is required to be 31 to construct an expander-like code. The 31-symbol shortened Reed Solomon codes have been chosen as the component code, with each symbol consisting of eight bits. Decoding algorithm is identical to Zemor's, except that if a particular vertex detects more errors than it can correct, it skips the decoding for that vertex. This is because as a side output, it is possible to compute using Berlekamp-Massey's algorithm for RS decoding, whether the degree of errors in the current input block of symbols to the decoder be corrected or not. If not, then the algorithm can be made to skip decoding, thus preserving the errors in the input block. This variation in decoding will reduce the number of extra errors introduced by that vertex if the decoding fails. Based on this decoding algorithm, a MATLAB model of decoder was first made, to observe code's performance. It uses the built-in RS decoding and encoding functions from MATLAB.

In yet another embodiment of the invention performance of the above code against random symbol errors is benchmarked. To benchmark the error-correction performance in the wake of

random errors, a decoder model for the proposed expander code was simulated in MATLAB and tested for different minimum distance values(Є) for the component codes. Symbol errors with different magnitudes were introduced at random locations of the all zero codevector. Convergence of the decoder's output back to the all zero codevector was checked at the simulation. As the code is linear, the performance obtained in testing for all zero codevector is valid for the entire code. Since the errors were introduced at random locations, simulations were run over many different rounds of decoding for different pseudo-random sequences as inputs, and averaged, to get reliable results. These sequences differ in random positions in which the errors are introduced. Each round of decoding for particular input further involves several iterations of execution of decoding algorithm. One iteration of decoding corresponds to both sides of the bipartite graph to finish decoding the component codes.

It is observed experimentally that in case of a decoding failure, beyond 4 iterations, the residual error in the overall codeword ceases to converge, implying that if correctable error patterns were present they would have been corrected within 4 iterations. In the case of correctable error patterns, residual errors after each iteration reduces and by the end of 4 iterations it would have converged to zero.

Hence the stopping threshold of decoder is fixed to exactly 4 iterations. The results of simulations are presented in **Table 1 and Table 2**. The component codes used for these simulations have minimum distance of 5 and 7 (symbols), respectively. The "failures" column represents the percentage of decoding failures. The "average number of iterations" column signifies the average number of iterations required for successful decoding of a corrupt codeword, over various rounds.

No. of errors	Failures	Avg. No. of iterations
50	0	1
80	1	1.71
100	18	2.33
110	40	2.72

Table 1 Random errors (Є= 5)

No. of errors	Failures	Avg. No. of iterations
150	0	1.6
175	0	1.99
200	0	2.19
250	23	3.82
275	64	4.5

Table 2 Random errors (Є= 7)

Some worst-case bounds on rate and error-correction capability of the proposed codes are presented in Table 3. The minimum distance of subcode has been varied between 3 and 15. Beyond 15, rate of the overall code C becomes very less and hence impractical. These results were compared with the theoretical bounds established by Zemor. Zemor proved that if $\epsilon \geq 3\lambda$, the total fraction of errors that can be corrected using the above algorithm is $\beta \frac{\epsilon}{2d}\left(\frac{\epsilon-2\lambda}{2d}\right)$ for_β < 1, where ∈ is the minimum distance, and *d* is the length of the subcode. This fraction can be seen proportional to $\frac{\epsilon^2}{4}$, an twelve-fold improvement over bound stated in Theorem 3. For calculating the Zemor bounds and making a fair comparison, it was needed to remove the advantage of using Reed Solomon codes as subcodes. Zemor had derived the bounds for general codes assuming that ≥ Є/ 2 errors could not be corrected for any distance (even/odd). For Reed Solomon component codes used in construction, since only odd distances have been used, (Є+ 1)/ 2 errors can never be corrected. To account for
this Є/2 has been replaced by (Є + 1)/2 in Zemor's formula to calculate the bounds.

Minimum distance of subcode (ϵ)	Subcode rate	Lower bound on rate of $\mathbb{C}$	Error-correcting capability of $\mathbb{C}$	Zemor's bound for $\mathbb{C}$
3	0.94	0.87	3	–
5	0.87	0.74	8	–
7	0.81	0.61	15	–
9	0.74	0.48	24	–
11	0.68	0.35	35	–
13	0.61	0.23	48	42
15	0.55	0.1	87	65

Table 3 Change in parameters of C with variation in minimum distance of subcode

A geometrical analysis of process of error correction in the overall code C has been given. Results from this analysis have also been used to derive the bounds on error correction capability of C. To do this analysis we observe that in P(5,GF(2)), points form the 0-dimensional subspaces and hyperplanes form the 4-dimensional subspaces (m = 0 and d-m-1 = 4). Also there are 7 points contained in a plane (2-dimensional subspace) and a plane is contained in 7 hyperplanes in P(5,GF(2)).

The analysis tries to answer the question: given the **minimum distance** Єof the subcode, what is the minimum number of random errors the code C can correct This is decided by the error pattern which causes the vertices corresponding to the points and hyperplanes of the graph get locked in such a way that in each iteration, an equal number of constraints fail on each side. This is the

minimal configuration of failure. Errors can expand over iterations (more edges representing symbols get corrupted), but that may not lead to the minimum configuration of failure. Similarly, if errors shrink, i.e. lesser number of vertices in bipartite graph fail in next iteration, then it leads to a case of decoding convergence, not decoding failure.

For example, if we consider Є= 5, each vertex of the graph can detect and correct up to 2 erroneous symbols ((Є-1)/ 2) in the set of symbols that it is decoding. If 3 or more erroneous symbols are given to it, then either the decoder, based on Berlekamp-Massey's algorithm, skips decoding, or it outputs another codeword that in worst case has at least Єdifferent symbols now (than the transmitted codeword), and hence at least Єerrors. However, the extra errors may get corrected if they do not lead to a minimal configuration of failure, and hence the invention does not concentrate on this case. So, if a case can be generated in which decoding of the subcode fails at vertices corresponding to 3 points, all of which are incident on 3 hyperplanes, a situation in which the 3 points will transfer at least 3 errors to each of vertices corresponding to the 3 hyperplanes. These vertices may also fail, or decode a different codeword, while decoding their inputs. Again in the worst case each of these hyperplane decoders will output at least 3 erroneous symbols. These corrupt symbols, or errors, are then transferred back to the vertices representing the 3 points. Thus, the errors will keep oscillating infinitely from one side of the graph to the other, and the decoder will never decode the right codeword. Thus, a minimum of 3*3=9 errors are required to cause a failure of decoding. We also assume in the worst case scenario that the "wrong" decoding by a vertex which is part of the minimum configuration of failure, does not reduce the number of errors. With this assumption, the bounds may be considered to be **"tight"**.

For any case in which less than 9 corrupt symbols exist, by pigeonhole principle, it will have at least one hyperplane or point having less than 3 errors incident on it. Decoder corresponding to that vertex will correctly decode the sub-code, thus reducing the total number of errors flowing in the overall decoder system of C. This will, in next iteration, cause some other hyperplane or point to have less than 3 errors. Thus in the subsequent iterations, all the errors will definitely be removed. Therefore, 8 errors or less will always be corrected. As it can be seen from **Table 1**, the worst case scenario is very unlikely to occur and for randomly placed errors, even 80 errors are found to be corrected 99% of the time.

Now the question is, when can one find a configuration in which 3 points are all incident on 3 hyperplanes? If any plane in the given geometry is chosen, any three points of that plane can be picked up and any 3 hyperplanes corresponding to the same plane can be found out. This will ensure that all the 3 points are incident on all the 3 hyperplanes. Thus, if for some input, the

decoding at these 3 points fails; in the worst case they will corrupt the entire edges incident on them. This in turn would cause 3 errors each, on the chosen hyperplanes. Hence the errors would oscillate between points and hyperplanes for each successive iteration. Thus, in the worst case, there need to be 9 erroneous symbols, located such that they are incident on the 3 chosen points, to cause the decoder of C to fail.

In general, if we are given a minimum distance Єof the subcode, it is known that at each vertex, more than ((Є+ 1)/ 2) errors will not be corrected. So, in the graph G we have to find the minimum number of vertices '**ξ'** required to get an embedded bipartite subgraph such that each vertex in the subgraph has a degree of at least ((Є+ 1)/ 2) towards vertices on other side of the subgraph. Once this number of vertices in some embedded subgraph has been found, the number of errors that can always be corrected by the proposed decoder is given by:

$$E = \xi\left(\frac{\epsilon + 1}{2}\right) - 1$$

In P(5,GF(2)), a plane has 7 points, and is contained in 7 hyperplanes. For 3 ≤ Є≤ 13, Єbeing odd, the minimum number of vertices **ξ** corresponds to ((Є+ 1)/ 2), and the corresponding points and hyperplanes can be picked from any plane. For Є**≥** 15, the calculation of **ξ** is non-trivial, since points and hyperplanes from one plane are not sufficient. This is because ((Є+ 1)/ 2) = 8, which would require us to get a subgraph of minimum degree 8. Construction of such embedded subgraph is not possible by choosing only one plane. A proof has been constructed to show non-existence of a minimum degree 8 embedded bipartite subgraph having order of 9 and 10, within point-hyperplane graph of P(5,GF(2)).

At a conjecture level, we suspect that an order-11, minimum degree 8 subgraph embedding is also not possible. Without using this conjecture, still, the lower bound of 87 as number of erroneous symbols in a 1953-length block of input based on C, as in the **Table 3**, is a loose lower bound. Since invented constructions are exact, these tight lower bounds can be used for all practical values of Є, wherever calculation of it is possible. Otherwise, another looser, lower bound can be found using eigenvalue calculations. Without that, it is still clear that the bound is much better than the bound obtained by Zemor using eigenvalue arguments.

In still another embodiment of the invention, performance of the code for burst errors is benchmarked. The strongest applications for this code lie in the areas of mass data storage such as discs. As pointed earlier, burst errors are the dominant cause of data corruption. Hence, the burst error correction capabilities of invented code have been examined. The burst error correction capability has been benchmarked against those codes designed in ECMA-130

standard for CD-ROM encoding, which is considered to be very robust to burst errors. In bipartite graph G constructed from P(5,GF(2)), the edges are labeled with integers, to map various symbols of the codeword. Such a labeling is not required to understand/ characterize the random error correction capability of the code. But here, the edges are labeled with numbers to try to maximize the burst error correction capability. This is achieved if each consecutive symbol, possibly part of a burst, is mapped to edges that are incident on distinct vertices representing different component decoders. Thus, consecutively numbered edges, representing consecutively located symbols in input symbol stream, go to different vertices hosting different RS decoders.

Since there are 63 vertices on one side of the graph, this scheme of numbering implies that edges incident on vertex 1 are assigned the numbers {1, 64, . . . , 1890}. Similarly, the edges incident on Vertex 2 are assigned {2, 65, . . . , 1891}, and so on. This numbering essentially achieves the effect of interleaving of code symbols. If the error correcting capacity of each component RS decoder is μ(=(Є-1)/ 2, for odd Є), then the minimum burst error correcting capacity of C will be μ*63. For example, for Є= 5, μ is 2, and the minimum burst error correcting capacity is 2*63 = 126. **Table 4** gives MATLAB simulation results for burst error correction for Є= 5.

No. of errors	Failures	Avg. No. of iterations
126	0	1
135	26	2.43

Table 4 Burst errors (Є= 5)

To demonstrate the excellent burst error correction capacity of the invented code, it has been benchmarked against the massive interleaving based codes in CD-ROMs. Traditionally in ECMA-130, the encoding utilizes heavy interleaving and dependence on erasure correction to deal with burst errors. For erasure correction, one level of decoding identifies the possible locations of the error symbols. The next level of decoding uses this information to correct them. The stage/process of interleaving used in CD-ROMs makes the encoding and decoding slower.
Two schemes have been proposed, which are described later in description, which offer significant improvement in burst error correction at similar data rates. The decoder, being fully parallel in its decoding, can handle larger sets of data at a time and hence could be used to increase the throughput. In the proposed schemes, however, it was required to fit the decoders in place of the heavy interleaving stage of the CD-ROM decoding data path, which only processes one frame at a time. Thus, in terms of throughput the CD-ROMs will be matching the proposed code in complexity but in burst error correction capability it will surpass them.

BEST MODE/EXAMPLE OF WORKING OF THE INVENTION

The invention is described in the example given below which is provided only to illustrate the invention and therefore should not be construed to limit the scope of the invention.

Multiple decoders based on expander-like codes described in the invention may be used to have two new decoding schemes that give better performance than the existing decoders. The details for such decoder for new, expander-like codes targeted for CD-ROM application is described below. The bipartite graph is constructed by using the point-hyperplane incidence relations of P(5,GF(2)). The points are generated using a primitive polynomial, which give the tapping points in a linear shift feedback register (LFSR). The primitive polynomial used to generate $GF(2^6)$ is x^6+x+1. The points of this projective space are given in **Table 5**. To identify the points lying on a particular hyperplane, first one has to construct the 5-dimensional vector subspace of the 6-dimensional vector space $GF(2^6)$. Then the points that correspond to vectors lying in that subspace can be taken as the points on the particular hyperplane. One such hyperplane, represented by its point set, is (0, 1, 2, 3, 4, 6, 7, 8, 9, 12, 13, 14, 16, 18, 19, 24, 26, 27, 28, 32, 33, 35, 36, 38, 41, 45, 48, 49, 52, 54, 56). The remaining 62 hyperplanes can be obtained by applying shift automorphism to this hyperplane.

Let the hyperplanes, numbered from 0 to 62, form one side, called A, of the bipartite graph between points and hyperplanes. Similarly, let the points, numbered from 63 to 125, form another side of the graph, called B. Based on this numbering, **Table 6** gives the complete list of the hyperplanes and their adjacent vertices. To improve the burst error capability, the edges need to be numbered such that consecutive edges always go to different vertices. Hence we label edge between vertex 0 and 63 as edge number 1, between vertex 1 and 64 as edge number 2 and so on.

Viewed as a computation graph, every vertex of this graph maps to a RS decoding computation. The input symbols to each of these decoders correspond to the edges which are incident to the vertex in question. During decoding, these symbols are provided as inputs in a specific order: message symbols first and then parity symbols. This order also gets reflected in the reduced row echelon form of the corresponding generator matrix, G. The edges incident on a vertex of side A (say, V1) are sorted with respect to increasing index numbers of the vertices reached on side B. This is the order in which the corresponding symbols are fed to the RS decoder represented by V1. A similar strategy is used for ordering inputs for RS decoders of vertices on side B.

While generating the bipartite graph and its edge/vertex labels, alternate representation described in **equation 1** is preferred. To recall, the points of an n-dimensional projective space over a field F can be taken to be the equivalence classes of nonzero vectors in the (n + 1)-dimensional vector space over F. Vectors in an equivalence class are all scalar multiples of one-another. These vector being one-dimensional subspaces, they also represent the rays of a vector space passing through origin. The orthogonal subspace of each such ray is the unique n-dimensional subspace of F^{n+1}, known as hyperplane. Each vector h of such orthogonal subspace is linked to the ray, p, by a dot product (addition is modulo 2 because of GF(2)) as follows.

$$p_0 h_0 + p_1 h_1 + \ldots + p_n h_n = 0$$

Where p_i is the i^{th} coordinate of p. This uniqueness implies bijection, and hence a vector p can be used to represent a hyperplane subspace, which is exclusive of this vector as a point. Due to duality, similar thing can be said about a hyperplane subspace. It is important to note that the above equation does not imply orthogonality. It is just a way to generate the hyperplane and point subspaces and is a convenient representation of incidence. Using this representation, vectors representing the hyperplanes can be found out, containing a given set of points, and then correlates them to the decimal numbers used to represent the hyperplanes. The representation of points and hyperplanes is further dependent on the representation of the underlying vector space. For the vector space, canonical representation is used. In this representation, those positions are set in a vector as 1 which corresponds to that power of x existing in the 1-D subspace. For example, point 0 is represented by 000001, point 2 as 000100, and point 8 as 001100 and so on. This representation can also easily be derived from **Table 5**.

Index	1-D subspace
0	{0,1}
1	$\{0, x^1\}$
2	$\{0, x^2\}$
3	$\{0, x^3\}$
4	$\{0, x^4\}$
5	$\{0, x^5\}$
6	$\{0, x^1 + 1\}$
7	$\{0, x^2 + x\}$
8	$\{0, x^3 + x^2\}$
9	$\{0, x^4 + x^3\}$
10	$\{0, x^5 + x^4\}$
11	$\{0, x^5 + x^1 + 1\}$
12	$\{0, x^2 + 1\}$
13	$\{0, x^3 + x\}$
14	$\{0, x^4 + x^2\}$
15	$\{0, x^5 + x^3\}$
16	$\{0, x^4 + x^1 + 1\}$
17	$\{0, x^5 + x^3 + x^2\}$
18	$\{0, x^3 + x^2 + x + 1\}$
19	$\{0, x^4 + x^3 + x^2 + x\}$
20	$\{0, x^5 + x^4 + x^3 + x^2\}$
21	$\{0, x^5 + x^4 + x^3 + x + 1\}$
22	$\{0, x^5 + x^4 + x^2 + 1\}$
23	$\{0, x^5 + x^3 + 1\}$
24	$\{0, x^4 + 1\}$
25	$\{0, x^5 + x^1\}$
26	$\{0, x^2 + x^1 + 1\}$
27	$\{0, x^3 + x^2 + x^1\}$
28	$\{0, x^4 + x^3 + x^2\}$
29	$\{0, x^5 + x^4 + x^3\}$
30	$\{0, x^5 + x^4 + x + 1\}$
31	$\{0, x^5 + x^2 + 1\}$

Index	1-D subspace
32	$\{0, x^3 + 1\}$
33	$\{0, x^4 + x\}$
34	$\{0, x^5 + x^2\}$
35	$\{0, x^3 + x + 1\}$
36	$\{0, x^4 + x^2 + x\}$
37	$\{0, x^5 + x^3 + x^2\}$
38	$\{0, x^4 + x^3 + x + 1\}$
39	$\{0, x^5 + x^4 + x^2 + x\}$
40	$\{0, x^5 + x^3 + x^2 + x + 1\}$
41	$\{0, x^4 + x^3 + x^2 + 1\}$
42	$\{0, x^5 + x^4 + x^3 + x\}$
43	$\{0, x^5 + x^4 + x^2 + x + 1\}$
44	$\{0, x^5 + x^3 + x^2 + 1\}$
45	$\{0, x^4 + x^3 + 1\}$
46	$\{0, x^5 + x^4 + x\}$
47	$\{0, x^5 + x^2 + x + 1\}$
48	$\{0, x^3 + x^2 + 1\}$
49	$\{0, x^4 + x^3 + x\}$
50	$\{0, x^5 + x^4 + x^2\}$
51	$\{0, x^5 + x^3 + x + 1\}$
52	$\{0, x^4 + x^2 + x\}$
53	$\{0, x^5 + x^3 + x\}$
54	$\{0, x^4 + x^2 + x + 1\}$
55	$\{0, x^5 + x^3 + x^2 + x\}$
56	$\{0, x^4 + x^3 + x^2 + x + 1\}$
57	$\{0, x^5 + x^4 + x^3 + x^2 + x\}$
58	$\{0, x^5 + x^4 + x^3 + x^2 + x + 1\}$
59	$\{0, x^5 + x^4 + x^3 + x^2 + 1\}$
60	$\{0, x^5 + x^4 + x^3 + 1\}$
61	$\{0, x^5 + x^4 + 1\}$
62	$\{0, x^5 + 1\}$

Table 5 Points of P(5, GF(2))

Hyperplane	Adjacent Vertices
0	63, 64, 65, 66, 67, 69, 70, 71, 72, 75, 76, 77, 79, 81, 82, 87, 89, 90, 91, 95, 96, 98, 99, 101, 104, 108, 111, 112, 115, 117, 119
1	64, 65, 66, 67, 68, 70, 71, 72, 73, 76, 77, 78, 80, 82, 83, 88, 90, 91, 92, 96, 97, 99, 100, 102, 105, 109, 112, 113, 116, 118, 120
2	65, 66, 67, 68, 69, 71, 72, 73, 74, 77, 78, 79, 81, 83, 84, 89, 91, 92, 93, 97, 98, 100, 101, 103, 106, 110, 113, 114, 117, 119, 121
3	66, 67, 68, 69, 70, 72, 73, 74, 75, 78, 79, 80, 82, 84, 85, 90, 92, 93, 94, 98, 99, 101, 102, 104, 107, 111, 114, 115, 118, 120, 122
4	67, 68, 69, 70, 71, 73, 74, 75, 76, 79, 80, 81, 83, 85, 86, 91, 93, 94, 95, 99, 100, 102, 103, 105, 108, 112, 115, 116, 119, 121, 123
5	68, 69, 70, 71, 72, 74, 75, 76, 77, 80, 81, 82, 84, 86, 87, 92, 94, 95, 96, 100, 101, 103, 104, 106, 109, 113, 116, 117, 120, 122, 124
6	69, 70, 71, 72, 73, 75, 76, 77, 78, 81, 82, 83, 85, 87, 88, 93, 95, 96, 97, 101, 102, 104, 105, 107, 110, 114, 117, 118, 121, 123, 125
7	70, 71, 72, 73, 74, 76, 77, 78, 79, 82, 83, 84, 86, 88, 89, 94, 96, 97, 98, 102, 103, 105, 106, 108, 111, 115, 118, 119, 122, 124, 63
8	71, 72, 73, 74, 75, 77, 78, 79, 80, 83, 84, 85, 87, 89, 90, 95, 97, 98, 99, 103, 104, 106, 107, 109, 112, 116, 119, 120, 123, 125, 64
9	72, 73, 74, 75, 76, 78, 79, 80, 81, 84, 85, 86, 88, 90, 91, 96, 98, 99, 100, 104, 105, 107, 108, 110, 113, 117, 120, 121, 124, 63, 65
10	73, 74, 75, 76, 77, 79, 80, 81, 82, 85, 86, 87, 89, 91, 92, 97, 99, 100, 101, 105, 106, 108, 109, 111, 114, 118, 121, 122, 125, 64, 66
11	74, 75, 76, 77, 78, 80, 81, 82, 83, 86, 87, 88, 90, 92, 93, 98, 100, 101, 102, 106, 107, 109, 110, 112, 115, 119, 122, 123, 63, 65, 67
12	75, 76, 77, 78, 79, 81, 82, 83, 84, 87, 88, 89, 91, 93, 94, 99, 101, 102, 103, 107, 108, 110, 111, 113, 116, 120, 123, 124, 64, 66, 68
13	76, 77, 78, 79, 80, 82, 83, 84, 85, 88, 89, 90, 92, 94, 95, 100, 102, 103, 104, 108, 109, 111, 112, 114, 117, 121, 124, 125, 65, 67, 69
14	77, 78, 79, 80, 81, 83, 84, 85, 86, 89, 90, 91, 93, 95, 96, 101, 103, 104, 105, 109, 110, 112, 113, 115, 118, 122, 125, 63, 66, 68, 70
15	78, 79, 80, 81, 82, 84, 85, 86, 87, 90, 91, 92, 94, 96, 97, 102, 104, 105, 106, 110, 111, 113, 114, 116, 119, 123, 63, 64, 67, 69, 71

Continued on next page

Hyperplane	Adjacent Vertices
16	79, 80, 81, 82, 83, 85, 86, 87, 88, 91, 92, 93, 95, 97, 98, 103, 105, 106, 107, 111, 112, 114, 115, 117, 120, 124, 64, 65, 68, 70, 72
17	80, 81, 82, 83, 84, 86, 87, 88, 89, 92, 93, 94, 96, 98, 99, 104, 106, 107, 108, 112, 113, 115, 116, 118, 121, 125, 65, 66, 69, 71, 73
18	81, 82, 83, 84, 85, 87, 88, 89, 90, 93, 94, 95, 97, 99, 100, 105, 107, 108, 109, 113, 114, 116, 117, 119, 122, 63, 66, 67, 70, 72, 74
19	82, 83, 84, 85, 86, 88, 89, 90, 91, 94, 95, 96, 98, 100, 101, 106, 108, 109, 110, 114, 115, 117, 118, 120, 123, 64, 67, 68, 71, 73, 75
20	83, 84, 85, 86, 87, 89, 90, 91, 92, 95, 96, 97, 99, 101, 102, 107, 109, 110, 111, 115, 116, 118, 119, 121, 124, 65, 68, 69, 72, 74, 76
21	84, 85, 86, 87, 88, 90, 91, 92, 93, 96, 97, 98, 100, 102, 103, 108, 110, 111, 112, 116, 117, 119, 120, 122, 125, 66, 69, 70, 73, 75, 77
22	85, 86, 87, 88, 89, 91, 92, 93, 94, 97, 98, 99, 101, 103, 104, 109, 111, 112, 113, 117, 118, 120, 121, 123, 63, 67, 70, 71, 74, 76, 78
23	86, 87, 88, 89, 90, 92, 93, 94, 95, 98, 99, 100, 102, 104, 105, 110, 112, 113, 114, 118, 119, 121, 122, 124, 64, 68, 71, 72, 75, 77, 79
24	87, 88, 89, 90, 91, 93, 94, 95, 96, 99, 100, 101, 103, 105, 106, 111, 113, 114, 115, 119, 120, 122, 123, 125, 65, 69, 72, 73, 76, 78, 80
25	88, 89, 90, 91, 92, 94, 95, 96, 97, 100, 101, 102, 104, 106, 107, 112, 114, 115, 116, 120, 121, 123, 124, 63, 66, 70, 73, 74, 77, 79, 81
26	89, 90, 91, 92, 93, 95, 96, 97, 98, 101, 102, 103, 105, 107, 108, 113, 115, 116, 117, 121, 122, 124, 125, 64, 67, 71, 74, 75, 78, 80, 82
27	90, 91, 92, 93, 94, 96, 97, 98, 99, 102, 103, 104, 106, 108, 109, 114, 116, 117, 118, 122, 123, 125, 63, 65, 68, 72, 75, 76, 79, 81, 83
28	91, 92, 93, 94, 95, 97, 98, 99, 100, 103, 104, 105, 107, 109, 110, 115, 117, 118, 119, 123, 124, 63, 64, 66, 69, 73, 76, 77, 80, 82, 84
29	92, 93, 94, 95, 96, 98, 99, 100, 101, 104, 105, 106, 108, 110, 111, 116, 118, 119, 120, 124, 125, 64, 65, 67, 70, 74, 77, 78, 81, 83, 85
30	93, 94, 95, 96, 97, 99, 100, 101, 102, 105, 106, 107, 109, 111, 112, 117, 119, 120, 121, 125, 63, 65, 66, 68, 71, 75, 78, 79, 82, 84, 86
31	94, 95, 96, 97, 98, 100, 101, 102, 103, 106, 107, 108, 110, 112, 113, 118, 120, 121, 122, 63, 64, 66, 67, 69, 72, 76, 79, 80, 83, 85, 87
32	95, 96, 97, 98, 99, 101, 102, 103, 104, 107, 108, 109, 111, 113, 114, 119, 121, 122, 123, 64, 65, 67, 68, 70, 73, 77, 80, 81, 84, 86, 88

Continued on next page

Hyperplane	Adjacent Vertices
33	96, 97, 98, 99, 100, 102, 103, 104, 105, 108, 109, 110, 112, 114, 115, 120, 122, 123, 124, 65, 66, 68, 69, 71, 74, 78, 81, 82, 85, 87, 89
34	97, 98, 99, 100, 101, 103, 104, 105, 106, 109, 110, 111, 113, 115, 116, 121, 123, 124, 125, 66, 67, 69, 70, 72, 75, 79, 82, 83, 86, 88, 90
35	98, 99, 100, 101, 102, 104, 105, 106, 107, 110, 111, 112, 114, 116, 117, 122, 124, 125, 63, 67, 68, 70, 71, 73, 76, 80, 83, 84, 87, 89, 91
36	99, 100, 101, 102, 103, 105, 106, 107, 108, 111, 112, 113, 115, 117, 118, 123, 125, 63, 64, 68, 69, 71, 72, 74, 77, 81, 84, 85, 88, 90, 92
37	100, 101, 102, 103, 104, 106, 107, 108, 109, 112, 113, 114, 116, 118, 119, 124, 63, 64, 65, 69, 70, 72, 73, 75, 78, 82, 85, 86, 89, 91, 93
38	101, 102, 103, 104, 105, 107, 108, 109, 110, 113, 114, 115, 117, 119, 120, 125, 64, 65, 66, 70, 71, 73, 74, 76, 79, 83, 86, 87, 90, 92, 94
39	102, 103, 104, 105, 106, 108, 109, 110, 111, 114, 115, 116, 118, 120, 121, 63, 65, 66, 67, 71, 72, 74, 75, 77, 80, 84, 87, 88, 91, 93, 95
40	103, 104, 105, 106, 107, 109, 110, 111, 112, 115, 116, 117, 119, 121, 122, 64, 66, 67, 68, 72, 73, 75, 76, 78, 81, 85, 88, 89, 92, 94, 96
41	104, 105, 106, 107, 108, 110, 111, 112, 113, 116, 117, 118, 120, 122, 123, 65, 67, 68, 69, 73, 74, 76, 77, 79, 82, 86, 89, 90, 93, 95, 97
42	105, 106, 107, 108, 109, 111, 112, 113, 114, 117, 118, 119, 121, 123, 124, 66, 68, 69, 70, 74, 75, 77, 78, 80, 83, 87, 90, 91, 94, 96, 98
43	106, 107, 108, 109, 110, 112, 113, 114, 115, 118, 119, 120, 122, 124, 125, 67, 69, 70, 71, 75, 76, 78, 79, 81, 84, 88, 91, 92, 95, 97, 99
44	107, 108, 109, 110, 111, 113, 114, 115, 116, 119, 120, 121, 123, 125, 63, 68, 70, 71, 72, 76, 77, 79, 80, 82, 85, 89, 92, 93, 96, 98, 100
45	108, 109, 110, 111, 112, 114, 115, 116, 117, 120, 121, 122, 124, 63, 64, 69, 71, 72, 73, 77, 78, 80, 81, 83, 86, 90, 93, 94, 97, 99, 101
46	109, 110, 111, 112, 113, 115, 116, 117, 118, 121, 122, 123, 125, 64, 65, 70, 72, 73, 74, 78, 79, 81, 82, 84, 87, 91, 94, 95, 98, 100, 102
47	110, 111, 112, 113, 114, 116, 117, 118, 119, 122, 123, 124, 63, 65, 66, 71, 73, 74, 75, 79, 80, 82, 83, 85, 88, 92, 95, 96, 99, 101, 103
48	111, 112, 113, 114, 115, 117, 118, 119, 120, 123, 124, 125, 64, 66, 67, 72, 74, 75, 76, 80, 81, 83, 84, 86, 89, 93, 96, 97, 100, 102, 104
49	112, 113, 114, 115, 116, 118, 119, 120, 121, 124, 125, 63, 65, 67, 68, 73, 75, 76, 77, 81, 82, 84, 85, 87, 90, 94, 97, 98, 101, 103, 105

Continued on next page

Hyperplane	Adjacent Vertices
50	113, 114, 115, 116, 117, 119, 120, 121, 122, 125, 63, 64, 66, 68, 69, 74, 76, 77, 78, 82, 83, 85, 86, 88, 91, 95, 98, 99, 102, 104, 106
51	114, 115, 116, 117, 118, 120, 121, 122, 123, 63, 64, 65, 67, 69, 70, 75, 77, 78, 79, 83, 84, 86, 87, 89, 92, 96, 99, 100, 103, 105, 107
52	115, 116, 117, 118, 119, 121, 122, 123, 124, 64, 65, 66, 68, 70, 71, 76, 78, 79, 80, 84, 85, 87, 88, 90, 93, 97, 100, 101, 104, 106, 108
53	116, 117, 118, 119, 120, 122, 123, 124, 125, 65, 66, 67, 69, 71, 72, 77, 79, 80, 81, 85, 86, 88, 89, 91, 94, 98, 101, 102, 105, 107, 109
54	117, 118, 119, 120, 121, 123, 124, 125, 63, 66, 67, 68, 70, 72, 73, 78, 80, 81, 82, 86, 87, 89, 90, 92, 95, 99, 102, 103, 106, 108, 110
55	118, 119, 120, 121, 122, 124, 125, 63, 64, 67, 68, 69, 71, 73, 74, 79, 81, 82, 83, 87, 88, 90, 91, 93, 96, 100, 103, 104, 107, 109, 111
56	119, 120, 121, 122, 123, 125, 63, 64, 65, 68, 69, 70, 72, 74, 75, 80, 82, 83, 84, 88, 89, 91, 92, 94, 97, 101, 104, 105, 108, 110, 112
57	120, 121, 122, 123, 124, 63, 64, 65, 66, 69, 70, 71, 73, 75, 76, 81, 83, 84, 85, 89, 90, 92, 93, 95, 98, 102, 105, 106, 109, 111, 113
58	121, 122, 123, 124, 125, 64, 65, 66, 67, 70, 71, 72, 74, 76, 77, 82, 84, 85, 86, 90, 91, 93, 94, 96, 99, 103, 106, 107, 110, 112, 114
59	122, 123, 124, 125, 63, 65, 66, 67, 68, 71, 72, 73, 75, 77, 78, 83, 85, 86, 87, 91, 92, 94, 95, 97, 100, 104, 107, 108, 111, 113, 115
60	123, 124, 125, 63, 64, 66, 67, 68, 69, 72, 73, 74, 76, 78, 79, 84, 86, 87, 88, 92, 93, 95, 96, 98, 101, 105, 108, 109, 112, 114, 116
61	124, 125, 63, 64, 65, 67, 68, 69, 70, 73, 74, 75, 77, 79, 80, 85, 87, 88, 89, 93, 94, 96, 97, 99, 102, 106, 109, 110, 113, 115, 117
62	125, 63, 64, 65, 66, 68, 69, 70, 71, 74, 75, 76, 78, 80, 81, 86, 88, 89, 90, 94, 95, 97, 98, 100, 103, 107, 110, 111, 114, 116, 118

Table 6 Point-Hyperplane Adjacency List

The burst error correcting capability of the envisaged code has been benchmarked against the code for CD-ROM error correction as detailed in ECMA-130 Standard. Based on this benchmarking, two novel schemes for CD-ROM encoding and decoding have been proposed. These schemes are based on the expander-like codes. Application of these codes, at various stages of CD-ROM encoding scheme and correspondingly in decoding scheme, substantially increases the burst error correcting capability of the disc drive.

The major part of error correction of the CD-ROM coding in ECMA-130 Standard scheme occurs in two stages, Reed-Solomon Product Code (RSPC) and Cross Interleaved Reed Solomon Code (CIRC). On the encoder side, Reed-Solomon Product Code (RSPC) stage comes before Cross Interleaved Reed Solomon Code (CIRC) stage, while on decoder side, Cross Interleaved Reed Solomon Code (CIRC) stage comes before Reed-Solomon Product Code (RSPC) stage.

To get an idea of the average error detection and correction capabilities of CDROM scheme(without erasures), Reed-Solomon Product Code (RSPC) and Cross Interleaved Reed Solomon Code (CIRC) stages of the ECMA standard were simulated in MATLAB. Because the CIRC uses delay elements to implement the interleaving, any frame arriving at the input of the second RS decoder has data symbols from the preceding 109 frames. Thus to gauge the error correction capabilities, it started with 218 frames of 32 symbols each. The 109th frame is considered as 0^{th} frame. After considering the effect of interleaving, this frame will require data symbols from the previous 109 frames. Errors are distributed over the last 109 frames and the error correction capability is observed. The details of Reed-Solomon Product Code (RSPC) and Cross Interleaved Reed Solomon Code (CIRC) stages are as follows.

Cross Interleaved Reed Solomon Code (CIRC): This stage leads to interleaving of codeword symbols. The massive interleaving done here is mainly responsible for the burst error correction. In a frame of 6976(=32*109*2) symbols, it can correct on an average 240 consecutive corrupt symbols. This amounts to approximately 2000 bits. If the burst is placed appropriately i.e. at the end of one frame of 6976 symbols and at the beginning of the next frame, then the Cross Interleaved Reed Solomon Code (CIRC) can potentially correct 4000 bits of burst errors. The success of error correction thus depends on the location of the burst.

Reed-Solomon Product Code (RSPC): After the Cross Interleaved Reed Solomon Code (CIRC) stage during decoding, some of the burst errors get corrected, and others get re-distributed among F1-Frames due to de-interleaving. Due to re-distribution and de-interleaving, the remaining errors can be considered as random errors. The Reed-Solomon Product Code (RSPC) stage in decoding then serves to correct these errors using Reed-Solomon decoding as erasure decoding. Success of this stage depends on the marking corrupt symbols as erasures by Cross Interleaved Reed Solomon Code (CIRC) stage.

Considering error detection and correction without erasures, Reed-Solomon Product Code (RSPC) and Cross Interleaved Reed Solomon Code (CIRC) on an average corrects a burst of 270 symbols in a frame of 6976 symbols. The scheme proposed here involves replacing one or both of the CIRC and the RSPC with encoders and decoders based on the expander-like codes. Burst error correction can be maximized in a way, without compromising on the code rate.

Scheme 1: As discussed, the Reed-Solomon Product Code (RSPC) and Cross Interleaved Reed Solomon Code (CIRC) subsystem in the decoder of ECMA-130 Standard can detect and correct a burst of about 270 erroneous symbols, in a frame of 32x218(=6976) 8-bit symbols. In the first scheme we propose, this subsystem is replaced by a set of 4 decoders, each of them being

expander-like code, C. The Reed Solomon subcodes used in C have block length d as 31(symbols). Further, we fix their minimum distance as Є = 7. The output of corresponding 4 encoders is further interleaved to improve performance, and de-interleaved on receiving side. Let k be the number of message symbols in each subcode. For Reed-Solomon code, which are maximum distance codes, we have n − k + 1 = Є, which implies that the code rate of the subcode is k/n = 25/31 = 0.806.

To construct these subcodes, a (255, 249, 7) Reed Solomon code has been chosen, The first 25 symbols are considered as message symbols; The remaining 224 symbols are set to 0. Hence we have a shortened RS code with each symbol (still) represented by 8 bits. Shortened Reed Solomon code has been used because each data symbol in the CD-ROM is a byte long.

For the overall code C, the code rate is equal to (2*r−1), where r is the rate of the (Reed Solomon) subcode. Hence the rate for codes used in each of the proposed encoders/ decoders is 2*0.806−1= 0.612. Thus, the number of message symbols for each decoder is equal to 1953* 0.612 = 1197. The rest are therefore parity symbols.

Cumulative input of the encoders and also the cumulative output of decoders, are set to a stream of 199 frames, each having 24 symbols payload. Assuming that each symbol can be encoded in 1 byte, this leads to generation of 4776 bytes. With 12 padding bytes added to it, this bigger set of 4788 bytes can be re-partitioned into 4 blocks of 1197 bytes each (4*1197=4788). Each block of 1197 source symbols can then be worked upon by 4 parallel working encoders. After encoding each block to 1953 symbols, one of the extra added (padding) bytes is removed from each encoder giving 244 frames of 32 bytes of data. Every RS decoder has Є =7, which implies that it can detect and correct up to 3 errors. Thus, each encoder for C will give a burst error correcting capability of 63*3=189. Since 4 of such encoders work in interleaved fashion, burst error detection and correction capability of at least 756 symbols among 244 frames can be achieved. This is against burst error correcting capability of 270 symbols in 218 frames, in the case of Reed-Solomon Product Code (RSPC)+Cross Interleaved Reed Solomon Code (CIRC) subsystem of ECMA-130 Standard

There are certain advantages associated with the scheme 1 such as, a massive improvement in burst error correction:270 in 32x218 symbols for CIRC+RSPC system, versus 756 in 32x244 in the proposed scheme and the code rate achieved is also comparable to the one for CIRC+RSPC subsystem. In the latter case, the code rate is 24/32=0.75, whereas in the proposed case it is 0.62.

There are certain disadvantages also, such as that this scheme is hardware expensive due to use of many parallel RS decoders. Also, the high throughput of the proposed decoder is not utilized. The resource complexity can be reduced by time-multiplexing the decoders, and also folding the architecture of each decoder..

Scheme 2: This scheme is a hardware economical scheme, which also increases the burst error correcting capability. Since the decoder also has a very good random error correcting capability, an error correction advantage can be achieved by replacing the Reed-Solomon Product Code (RSPC) stage of the ECMA-130 Standard with the envisaged encoder/decoder (with ϵ=5). Two of the encoders can replace the Reed-Solomon Product Code (RSPC) encoder in this scheme. Data from these encoders is then interleaved, and passed on to the Cross Interleaved Reed Solomon Code (CIRC). In the decoding stage, after Cross Interleaved Reed Solomon Code (CIRC) there is correspondingly de-interleaving and this is followed by decoding based on the invented code.

This scheme has the advantage that it increases the error correction capability. It also matches the code rate of CIRC: 0.75 for Cross Interleaved Reed Solomon Code (CIRC), versus 0.74 for the decoder. Also, it is a hardware economical scheme. MATLAB simulations shows that the burst error rate goes up from 270 for Reed-Solomon Product Code (RSPC)+Cross Interleaved Reed Solomon Code (CIRC) subsystem, to more than 400 for Cross Interleaved Reed Solomon Code (CIRC) and the encoder. **Table 7 and Table 8** show some simulation results for this scheme.

No. of errors	Failures
270	2
300	45
400	86

Table 7 Response to burst errors for Reed-Solomon Product Code (RSPC)+Cross Interleaved Reed Solomon Code (CIRC)

No. of errors	Failures
400	7
450	17
500	26

Table 8 Response to burst errors in Scheme 2

In this scheme, we use 2-error correcting RS decoders. The scheme could be hardware expensive scheme in terms of number of decoders used(126 in this case). However, the throughput of decoder based on this scheme is so high that we can fold computations to use fewer processors, and still do better than the ECMA data rate. FPGA prototyping of one decoder with $\epsilon=5$ and folded architecture gave throughput as 125 Mbyte/sec, whereas fastest commercial CD-ROM data rate has throughput of upto 11 Mbyte/sec (source: wikipedia). If implemented in ASIC, the throughput performance is only expected to improve.

Application to DVD-R

The Reed Solomon class of codes can also be applied to evolve encoding and decoding for DVD-ROM. This particular application of the new coding scheme also brings out the fact that taking a bipartite graph G from a higher-dimensional projective space can be advantageous in terms of better rate and better error correction capacity.

Present DVD-R Error correction blocks

The details of the ISO/IEC standard implementation have been known and an overview has been provided for the main error correcting block, which will be used to derive a new correcting scheme. The data received from the host, called Main Data, is formatted in a number of steps, before being recorded on the disk. It is transformed successively into following.

- A Data Frame,
- A Scrambled Frame,
- An ECC Block,
- A Recording Frame, and
- A Physical Sector.

Data Frames

A Data Frame consists of 2064 bytes arranged in an array of 12 rows, each containing 172 bytes. The first row starts with three fields spanning 12 bytes interval, which is followed by 160 Main Data bytes. The next 10 rows each contain 172 Main Data bytes. The last row contains 168 Main Data bytes, followed by four check bytes of Error Detection Code (EDC). Thus there are 2048 bytes of Main Data in each Data frame of size 2064.

Scrambled Frames

The 2048 Main Data bytes are later scrambled. Scrambling bytes are generated with a Linear Feedback Shift Register (LFSR), and each data byte is XOR-ed with a corresponding scrambling byte.

ECC Block

Referring to **Figure 4**, an ECC Block is formed by arranging 16 consecutive data frames, after scrambling, in an array of 192 rows (402) of 172 bytes (404) each. To each of the 172 columns, 16 bytes of parity of an outer code (406) are added. This results in a block having 208 rows having 172 bytes each. To each resulting 208 rows, 10 byte of parity of an inner code (408) is added. Thus a complete ECC Block comprises 208 rows of 182 bytes each. The bytes of this array are identified as $B_{i,j}$, where i is the row number and j the column number. Thus the ECC block is nothing but a Reed Solomon Product Code, with the inner code being a RS (182,172,11) code, and the outer code being a RS (208,192,17) code.

Recording Frames

Sixteen recording frames are obtained by partitioning an ECC block into 16 frames, while simultaneously doing interleaving. This is achieved by interleaving one of the 16 PO rows (406) as illustrated in **Figure 4,** at a time after every 12 rows of an ECC Block. **Figure 5** brings out this partitioning graphically. The rate of this encoding can simply be calculated as 2064*16/37856 = 0.8724.

Analysis of Error Correction in DVD-R

The main error correction in DVD-R is provided by the Reed-Solomon Product Code (RSPC) block, which consists of an inner RS (182,172,11) code and an outer RS (208,192,17) code. The inner code can detect and correct up to 5 errors ((11−1)/2), while the outer code can detect and correct 8 errors.

Random Errors

The minimum distance of the overall code is 17*11=187. Hence in a block of (208*182)=37856 bytes, the number of random errors should be less than or equal to (187−1)/2=93, in order that the block be decoded completely. Further it has been pointed out that the number of errors in 8 consecutive ECC blocks must be less than or equal to 280, for correct decoding.

Burst Errors

The Reed-Solomon Product Code (RSPC) used in DVDs is very robust towards burst errors. Since the data is arranged in a matrix fashion, as it can be seen in **Figure 4**, it is easy to calculate the worst case amount of errors than can be corrected by the inner and outer codes.

If erasures are not considered, then the inner code can correct a burst of 5 errors, while the outer code can correct a burst of 8 errors. The biggest burst of errors that we can be corrected in the product code can be derived as follows.

• 8 rows of 182 bytes can be allowed to get completely corrupted. Then, these will be detected and corrected by the outer code's decoders, which operate on columns of the matrix.

• Further, an additional 5 bytes can be allowed to get corrupted in the row above and below the set of 8 rows. Since inner code decoding happens first, these errors will be corrected by the inner code's decoders, after which the outer code's decoders can correct the remaining 8 rows of contiguous corrupted data.

It is clear that without considering erasure decoding, 8*182+5*2=1466 errors can still be still detected and corrected in a burst, in one block of 37856 bytes. If two ECC blocks are placed back to back, the number of errors that can be detected and corrected increases. Another 8 rows of 182 corrupt bytes can be added in ECC block 1, 5 bytes before these 8 rows (corrected by inner codes), 8 rows of 182 corrupt bytes in ECC block 2 and 5 bytes after these 8 rows. This gives a total of 8*2*182+5*2=2922 errors' burst that can be corrected simultaneously.

If the inner decoding is allowed to mark as erasures, all the bytes of a codeword that has more errors that it can correct, the overall burst error correction can be increased even further. With only erasures, the outer code can correct 16 erasure symbols. Thus the total error correction capacity will be 16*2*182+5*2=5834 bytes. This is the absolute maximum burst that this stage can handle.

A New ECC Scheme for DVD-R

A scheme presented here is based on expander-like codes, C. This scheme, which uses decoders for code C, can improve the error correction capacity of the DVD. In this scheme, the Reed-Solomon Product Code (RSPC) stage of the DVD encoding is substituted by encoders of code C.

The encoders are therefore employed during the transformation of Data Frames into Recording Frames. In order to be compliant with the rest of the standard, the encoded data must be taken as output as a block having same size as one recording frame, i.e. 2366(=13*182) bytes. Thus, 2064 bytes per data frame has been taken as input, and need to output 2366 bytes in the format of recording frames.

Since higher rate and a better error correction capacity is required, there is a need to look at higher dimensions of PG. Higher dimensional of PG lead to better expansion properties, which leads to better error correction. In one such scheme, PG(8,GF(2)) is considered. This geometry contains $2^9-1=511$ points and hyperplanes. Further, each point is contained in $2^8-1=255$ hyperplanes and similarly, each hyperplane contains 255 points. Thus, the degree of each vertex in the bipartite graph constructed will be 255, and the number of vertices in each partition of the bipartite graph will be 511. Each edge of the graph represents an 8-bit symbol. Thus, the total number of symbols in the overall code G_8 will be 255*511=130305.

Each vertex of the graph is a Reed Solomon decoder which corresponds to a RS (255,239,17) code. Thus, each vertex can detect and correct 8 errors. The rate of the sub-code in this case is r=239/255 = 0.9373. The overall rate of the code is at least $2*r-1=0.8745$, which is marginally better than the rate in the ISO/IEC standard for DVD encoding. The overall burst error correction capability without erasures will be 8*511=4088 bytes, which is much greater than 2922. Since 2366*55=130130, 55 frames in one round of encoding can be taken as output. Hence 175 lesser source symbols have been chosen to encode, than required by G_8. These remaining 175 source symbols will be set to 0, a padding byte value. These 175 padding bytes are later dropped after encoding, to get an overall encoded block of 130130 bytes. By using a systematic encoding matrix, the locations of these padding bytes remain intact during the process of encoding. Because of using 175 lesser source symbols for encoding, the overall rate drops to 0.8732.

It is reasonable to assume pipelining while decoding, because of the application of DVDs in many real time applications like video etc. Thus, two frames of 130305 bytes put together can detect and correct a burst of 4088*2=8176 bytes. This number is quite bigger than the absolute maximum burst of 5834 brought out earlier. Thus, even if erasures are not considered, at a slightly higher rate, much better burst error correction capability is achieved.

Thus, at the price of extra hardware for decoding, and memory for storing the large frame sizes, a linear time decoder with an exceptional burst error correction capability can be obtained. As far as random errors are concerned, average case performance will easily surpass the existing standard. The existing standard specifies that the number of random errors in 8 consecutive ECC blocks must be less than or equal to 280. The frame size of 130305 corresponds to approximately 3.44 consecutive ECC blocks. Preliminary MATLAB simulation results show that around 1990 random errors are always corrected in one iteration of the decoding itself.

As a concluding note, the design of expander-like code to be used in DVD-R application is not limited to choice of RS (255,239,17) code. In fact, as the minimum distance increases from 17 to 21 (i.e. RS(255, 235, 21) code), a burst error correction 511*2 more than 4088 can be obtained, that is, 5110, while the drop in overall code rate is just 0.03. If 'r' expander-like encoders are used simultaneously, and their outputs interleaved, then also the error-correction capability goes up by factor of 'r'.

Application in Wireless Communication

These codes may be used to perform adaptive or selective encoding of data in wireless communication systems. Most digital wireless communication standards specify logical control channels that can provide feedback on channel conditions based on various radio measurements. One of the challenges of land mobile communications is the time varying channel conditions caused by attenuation, fading and other changes of propagation environment.

Hence, for example, if the SNR of channel is high, then only one set of RS encoders corresponding to a side in bipartite graph of expander-like construction can be used to do one-level encoding. Each RS encoder in such case works on a portion of block of inputs symbols. Sometime later, if and when SNR becomes poor, more powerful encoding can be done by using entire bipartite graph and RS code embedding into it, which has much better error correction capability. This way, the effect of adaptive multirate coding may be achieved.

The preceding description has been presented with reference to various embodiments of the invention. Persons skilled in the art and technology to which this invention pertains will appreciate that alterations and changes in the described structures and methods of operation can be practiced without meaningfully departing from the principle, spirit and scope of this invention.

Advantages of the invention

Choice of Projective Space

Extensive results have been given for the construction derived from P(5,GF(2)), but in no way it is restricted to this dimension. Major reason for picking up this dimension was that the degree of each vertex in the graph derived is 31, which is very close to the F2 frame size of 32 bytes in CD-ROMs. Thus, the choice of this dimension would help in benchmarking the performance of code against an industry standard.

Advantages of Using PG Graphs

If one looks only at the decoding algorithm and construction of the graph, it would seem that any simple, regular bipartite graph would be a good candidate for having good expander codes and corresponding decoders. Some of the reasons for using projective geometry based expander codes are as follows.

As discussed, the mapping of vertices to points and hyperplanes enables one to use several projective geometry properties for disproving the existence of certain bipartite subgraphs of a fixed minimum degree. This strategy leads to finding the minimum number of vertices required to form a complete bipartite subgraph of a given minimum degree. This number of vertices is required to calculate the bounds for error correction capability of the overall code. Thus, it is not needed to utilize the Zemor bound for computing the minimum distance . Also, the bounds obtained in this manner are better than the Zemor bound. Furthermore, Zemor had restricted the subcodes to be constrained by $d \geq 3\lambda$, λ being the second largest eigenvalue of the graph. The present invention has no such restriction.

The use of projective geometry also helps in developing a perfect folded architecture of the decoder for hardware implementation. Folded architectures enable efficient utilization of processors and memories, without any significant increase in scheduling complexity.

Again, as shown earlier, the probability of the errors getting locked in a subgraph during iterative decoding is very less. This translates to saying that probability of occurrence of infinite oscillations due to errors while decoding is extremely low .

Advantages of Expander-like Codes

This choice of higher dimension has some significant advantages over using 2-dimensional projective space for code construction.

1. The projective geometry based graphs that we have considered for designing the expander-like codes are good Ramanujan graphs. Ramanujan graphs are graphs that follow the second eigenvalue upper bound property: $\lambda_2 \leq 2\sqrt{d-1}$ where d is the degree of a regular graph. A tight lower bound on the expansion coefficient of balanced, regular bipartite graphs is provided as:

$$\left(1 - \frac{{\lambda_2}^2}{d^2}\right)$$

Where, λ_2 is the second largest eigenvalue of adjacency matrix of the bipartite graph. Since balanced, regular bipartite graphs are used, this lower bound holds for our graph. One can hence deduce from this bound that smaller the second eigenvalue is, higher the expansion coefficient consequently is, and hence better the expansion properties of the graph is. This in turn, indirectly from expansion properties or otherwise, leads to construction of good expander-like codes.

The importance of Ramanujan graphs for being considered as better expanders stems from an earlier result by Alon. In this result, Alon proves that for any arbitrary family of d-regular graphs (need not be Ramanujan graph), their second eigenvalue is asymptotically lower bounded with the same threshold: $2\sqrt{d-1}$ X Hence Ramanujan graphs as a particular subclass of d-regular graphs are important because they reach this threshold, thus having higher expansion coefficients, and hence are better expanders.

To establish PG as good Ramanujan graph, it has been shown that the second eigenvalue of the adjacency matrix of corresponding PG-based graph is small enough, than it upper bound. The second eigenvalue λ_2 of adjacency matrix made out of point-hyperplane incidence of a projective space can be derived as:

$$q^{\frac{t-1}{2}}$$

Where, q is the base prime and t is the dimension of the projective space. In the present invention, q = 2 and t = 5, hence $\lambda_2 = 2^{(t-1)/2} = 4$. The corresponding Ramanujan upper bound on
λ_2 is $2\sqrt{2^t - 1}$X .

Since the number of points incident on a hyperplane in a t-dimensional projective space is 2^t. This upper bound turns out to be 11.31 in our case, and ignoring the subtraction of 1, approximately $2^{(t+2)/2}$ in general. Now the actual value of λ_2 is sufficiently smaller than the upper bound in general; almost always by an approximate factor of $2\sqrt{2}9.\sqrt{9}$. Hence PG bipartite graphs are good Ramanujan graphs in general.

2. The usage of higher dimensional projective space than 2 leads to better expansion properties/coefficient. The lower bound on expansion coefficient of t-dimensional PG-based bipartite graph turns out to be $1 - 2^{-(t+1)}$. Hence as t increases(dimension of projective space), the lower bound on expansion coefficient asymptotically tends to 1, and hence perhaps using higher dimensional PG bipartite graph leads to construction of

better expanders. Similar advantages of using higher-dimensional projective space have been found in other applications of this coding scheme.

3. The decoding complexity of a parallel decoding in schedule based on expander-like codes has RS subcode decoding as the major contributor. If almost same (n,k,n-k+1) RS codes are chosen to compare expander-like codes, versus codes designed using n-dimensional projective space, the overall (decoding) computation complexity will be mainly determined by the block length of the code. If it is assumed that the RS subcode length used in present codes is (2^n-1), then the block length for codes is $(2^n-1)\times(2^{n+1}-1) = O(2^{2n}) = O(\text{subcode length}^2)$. For 2-dimensional projective space, the closest comparable subcode length is (2^n+1). In such a case, the block length becomes $(2^n + 1) \times (2^{2n} + 2^n + 1) = O(2^{3n}) = O(\text{subcode length}^3)$. The data in **Table 9** reinforces this point.

4. From Table 9, it is further clear that using similar-sized RS-decoders, expander-like codes constructed using higher-dimensional PG(where $n \neq 3$ and 4) have better error correction capability than those constructed using 2-d PG. During comparison, the length of RS subcodes has been kept similar (2^n -1 vs 2^n+1). For n = 3 and 4, having subcode rate > 0.5 is not possible, while simultaneously assuming subcode distance $\epsilon > 3 * \lambda$ (Zemor's decoding constraint). For n = 5 onwards, while the rates for 2-d PG based expander-like codes are better than their n-d counterparts by a factor of 2.5 or less, the (%) error correction capability(and hence minimum relative distance) is consistently better for the n-d counterparts (touching almost 90 times better). As a side note, this observation also brings out the classical tradeoff between rate and distance (relative distance) of any class of codes.

5. Since the degree of graph used are of nature (2^n -1), RS codes can be used which are not shortened. Whereas, in case of projective plane based graph codes, (2^n + 1) almost always leads to usage of shortened RS codes. Using shortened RS codes reduces the code rate of the subcode, and hence the expander construction as well.

6. Since each decoder is RS decoder, it can also be used in sense of optimum erasure code decoder, since RS codes are optimum erasure correction codes. Then, the burst error correction capability gets doubled for each decoder.

Length of RS Subcodes	7(n=3)	9(n=2)
	15(n=4)	17(n=2)
	31(n=5)	33(n=2)
	63(n=6)	65(n=2)
	127(n=7)	129(n=2)
	255(n=8)	257(n=2)
	511(n=9)	513(n=2)
	1023(n=10)	1025(n=2)
Order of Bipartite Graph	15(n=3)	73(n=2)
	31(n=4)	273(n=2)
	63(n=5)	1057(n=2)
	127(n=6)	4161(n=2)
	255(n=7)	16513(n=2)
	511(n=8)	65793(n=2)
	1023(n=9)	262657(n=2)
	2047(n=10)	1049601(n=2)
Length of Overall Code	105(n=3)	657(n=2)
	465(n=4)	4641(n=2)
	1953(n=5)	34881(n=2)
	8001(n=6)	270465(n=2)
	32385(n=7)	2130177(n=2)
	130305(n=8)	16908801(n=2)
	522753(n=9)	134743041(n=2)
	2094081(n=10)	1075841025(n=2)
Second Eigenvalues	2.000000(n=3)	1.414214(n=2)
	2.828427(n=4)	1.414214(n=2)
	4.000000(n=5)	1.414214(n=2)
	5.656854(n=6)	1.414214(n=2)
	8.000000(n=7)	1.414214(n=2)
	11.313708(n=8)	1.414214(n=2)
	16.000000(n=9)	1.414214(n=2)
	22.627417(n=10)	1.414214(n=2)
Zemor's Constraint based Subcode min distance	6(n=3)	5(n=2)
	9(n=4)	5(n=2)
	12(n=5)	5(n=2)
	17(n=6)	5(n=2)
	24(n=7)	5(n=2)
	34(n=8)	5(n=2)
	48(n=9)	5(n=2)
	68(n=10)	5(n=2)
Rate of Overall Code	-0.428571(n=3)	0.111111(n=2)
	-0.066667(n=4)	0.529412(n=2)
	0.290323(n=5)	0.757576(n=2)
	0.492063(n=6)	0.876923(n=2)
	0.637795(n=7)	0.937984(n=2)
	0.741176(n=8)	0.968872(n=2)
	0.816047(n=9)	0.984405(n=2)
	0.869013(n=10)	0.992195(n=2)
Maximum Errors Corrected	1(n=3)	7(n=2)
	1(n=4)	9(n=2)
	1(n=5)	11(n=2)
	1(n=6)	11(n=2)
	1(n=7)	12(n=2)
	2(n=8)	12(n=2)
	2(n=9)	12(n=2)
	2(n=10)	12(n=2)
Percentage Errors Corrected	0.952381%(n=3)	0.456621%(n=2)
	0.215054%(n=4)	0.086188%(n=2)
	0.051203%(n=5)	0.014334%(n=2)
	0.012498%(n=6)	0.001849%(n=2)
	0.003088%(n=7)	0.000282%(n=2)
	0.001535%(n=8)	0.000035%(n=2)
	0.000383%(n=9)	0.000004%(n=2)
	0.000096%(n=10)	0.000001%(n=2)

Table 9 Comparison of n-dimensional vs 2-dimensional PG-based Expander-like Codes

Choice of Reed-Solomon Code (RS Code)

The parameters of the RS code have been chosen to enable efficient schemes that could be applicable to data storage systems. In case of application to CD-ROM, since the smallest symbol size is a byte, RS code of size 255 has been used.

The degree of each vertex in the bipartite graph used there being 31, the RS code size has to be shorten to 31. This shortening is done by dropping the message symbols from 32 onwards. More specifically, if the minimum distance is ϵ, symbols 32 to $255 - \epsilon + 1$ are assumed to be zero.

In turn, the minimum distances are chosen to get the overall rate of code to match the rate used by applications such as the CD-ROM encoding scheme. Additional error correction capability is achieved by adding a small amount of interleaving.

WE CLAIM

1. A method for error control coding for a digital storage device or packet data transmission; the said method is characterized by use of a symbol error correcting code based on at least one higher dimensional projective geometry graph followed by mapping the symbols of the error correcting code to the edges of the said one or more graph, wherein the said higher dimensional projective geometry based graph is generated by the computer implemented steps of:

    a. defining the projective space wherein the said projective space of dimension d consists of one dimensional subspaces of a (d+1)-dimensional vector space and an m-dimensional subspace of the projective space consists of all one dimensional subspaces of a (m+1)-dimensional subspace of the vector space;

    b. deriving the higher dimensional projective geometry based graph from the incidence relations of said projective space, wherein the higher dimensional projective geometry based graph is having at least two partitions and each partition is having at least one vertex;

    c. associating one vertex of the graph with each m-dimensional subspace and one with each (d-m-1)-dimensional subspace of the said projective space wherein each vertex of the graph has a fixed degree and the number of vertices in each partition of the graph are equal;

    d. connecting all vertices from one partition with other vertices from the other partition by an edge if the corresponding subspaces are incident on each other; and

    e. decoding all the symbols by at least one vertex in one partition of the higher dimensional projective geometry based graph, wherein the symbols correspond to the edges incident on the vertex; and coding for error control in a communication and digital storage device by use of a symbol error correcting code based on at least one derived higher dimensional projective geometry graph.

2. A method as claimed in claim 1, wherein the higher dimensional projective geometry based graph is a bipartite graph.

3. A method as claimed in claim 1, wherein the error correcting code is an expander-like code having reed solomon (RS) code as component code.

4. A method as claimed in claim 1, the symbol error control coding is based on higher dimensional projective geometry based graph, wherein the higher dimensional projective geometry is at least two in numbers.

5. A method as claimed in claim 1, the symbol error correcting codes are provided for large data blocks in digital storage device or packet data transmission wherein the length of data blocks is at least one thousand symbols.

6. A method as claimed in claim 1, wherein the digital storage device is selected from the set of disc based secondary digital storage devices comprising RAID systems, HDD, CD-ROM and DVD-ROMs.

7. A method as claimed in claim 1, wherein the symbol error correcting code is employed to detect and correct erroneous symbols of at least one said digital storage device.

8. A method as claimed in claim 1, wherein the symbol error is selected from burst error or random error.

9. A system for error control coding in a digital storage device or packet data transmission; the said system comprising at least one decoding device, at least one digital storage device and at least one memory element communicatively coupled with each other, wherein the said system is characterized in using a symbol error correcting code based on at least one higher dimensional projective geometry graph followed by mapping the symbols of the error correcting code to the edges of the said one or more graph, wherein the said higher dimensional projective geometry based graph is generated by same steps as in claim 1.

10. A system as claimed in claim 9, wherein the said decoding device is a reed solomon (RS) decoder.

11. A system as claimed in claim 10, wherein the said reed solomon (RS) decoder is having ability to skip decoding if more than correctable errors are detected.

12. A system as claimed in claim 9, wherein the error correcting code is an expander-like code having reed solomon (RS) code as component code.

13. A system as claimed in claim 10, wherein the reed solomon (RS) decoder are capable of handling shortened reed solomon (RS) codes.

14. A system as claimed in claim 9, wherein the higher dimensional projective geometry based graph is a bipartite graph;

15. A system as claimed in claim 9, the symbol error control coding is based on higher dimensional projective geometry based graph, wherein the higher dimensional projective geometry is at least two in number.

16. A system as claimed in claim 9, the symbol error correcting codes are provided for large data blocks in digital storage device or packet data transmission wherein the length of data blocks is at least one thousand symbols.

17. A system as claimed in claim 9, wherein the digital storage device is selected from the set of disc based secondary digital storage devices comprising RAID systems, HDD, CD-ROM and DVD-ROMs.

18. A system as claimed in claim 9, wherein the symbol error correcting code is employed to correct at least one error of at least one said digital storage device.

19. A system as claimed in claim 9, wherein the symbol error is selected from burst error or random error.

20. A system as claimed in claim 9, wherein the system for symbol error control coding for digital storage device or packet data transmission has a parallel and symmetric design.

21. A system as claimed in claim 9, wherein the memory element is utilized to store the address space in the required order of computation.

22. A system and method substantially as herein described with reference to and as illustrated by the accompanying drawings.

Dated this 00th day of July, 2010

Priyank Gupta

Agent for Applicant

IN-PA-1454

ABSTRACT

Method and System for Error Control Coding Using Expander-like Codes

A method and system for error control coding using expander-like codes constructed from higher dimensional projective geometry based graphs is presented. The invention provides a method and system for error control coding which has exceptional random and burst error detection and correction capabilities for large data blocks in storage devices and in communication such as CD ROM, DVD ROM etc.

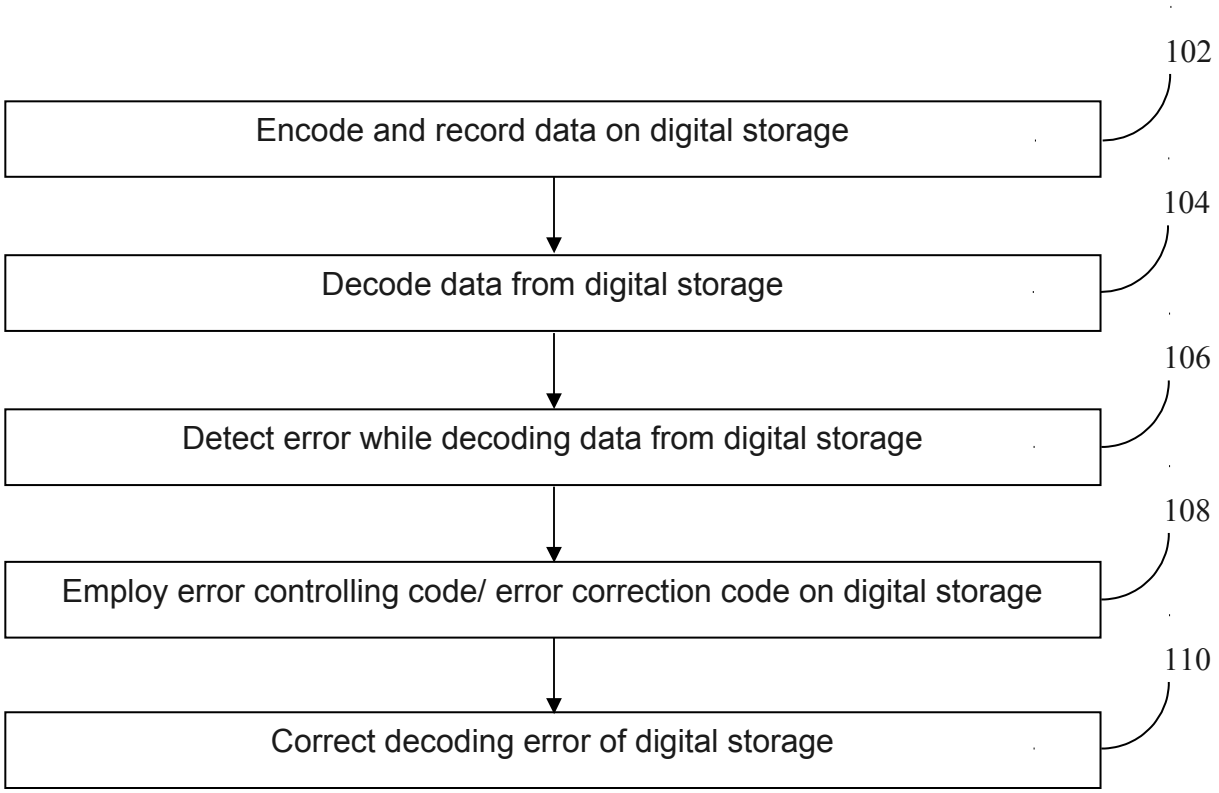

Figure 1

Prior art

Priyank Gupta

Agent for Applicant
IN-PA-1454

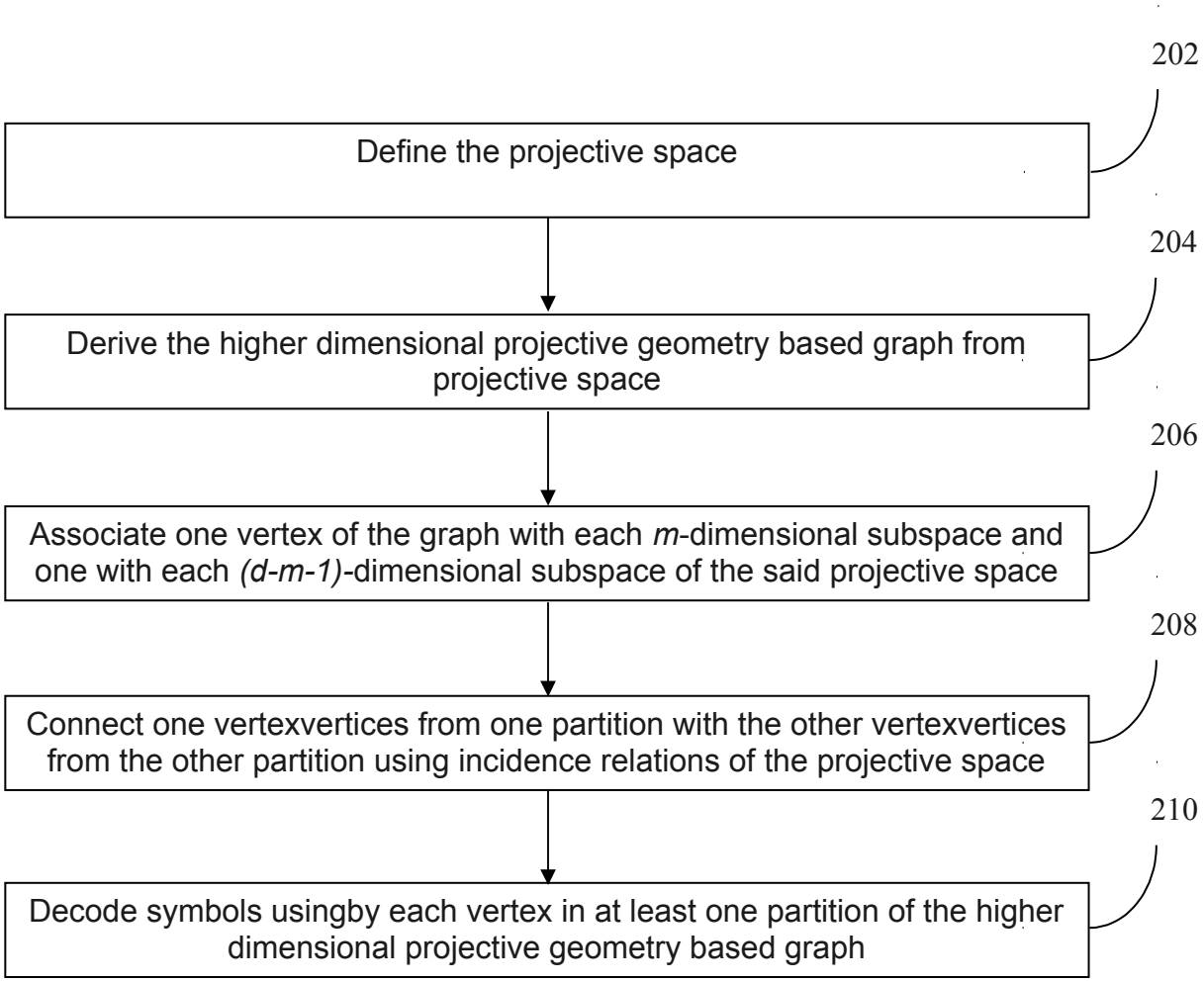

Figure 2

Priyank Gupta

Agent for Applicant
IN-PA-1454

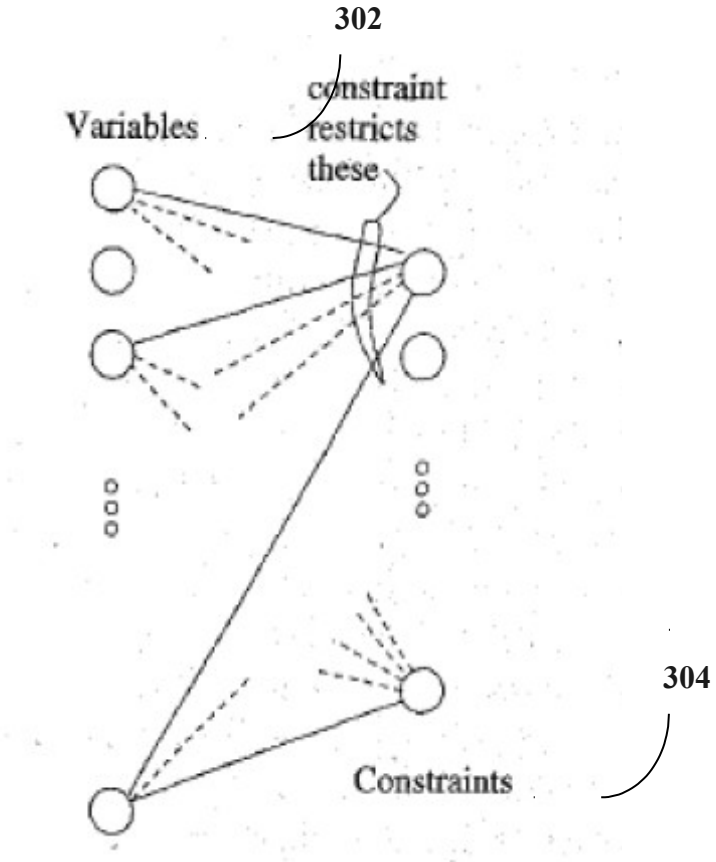

Figure 3

Priyank Gupta
Agent for Applicant
IN-PA-1454

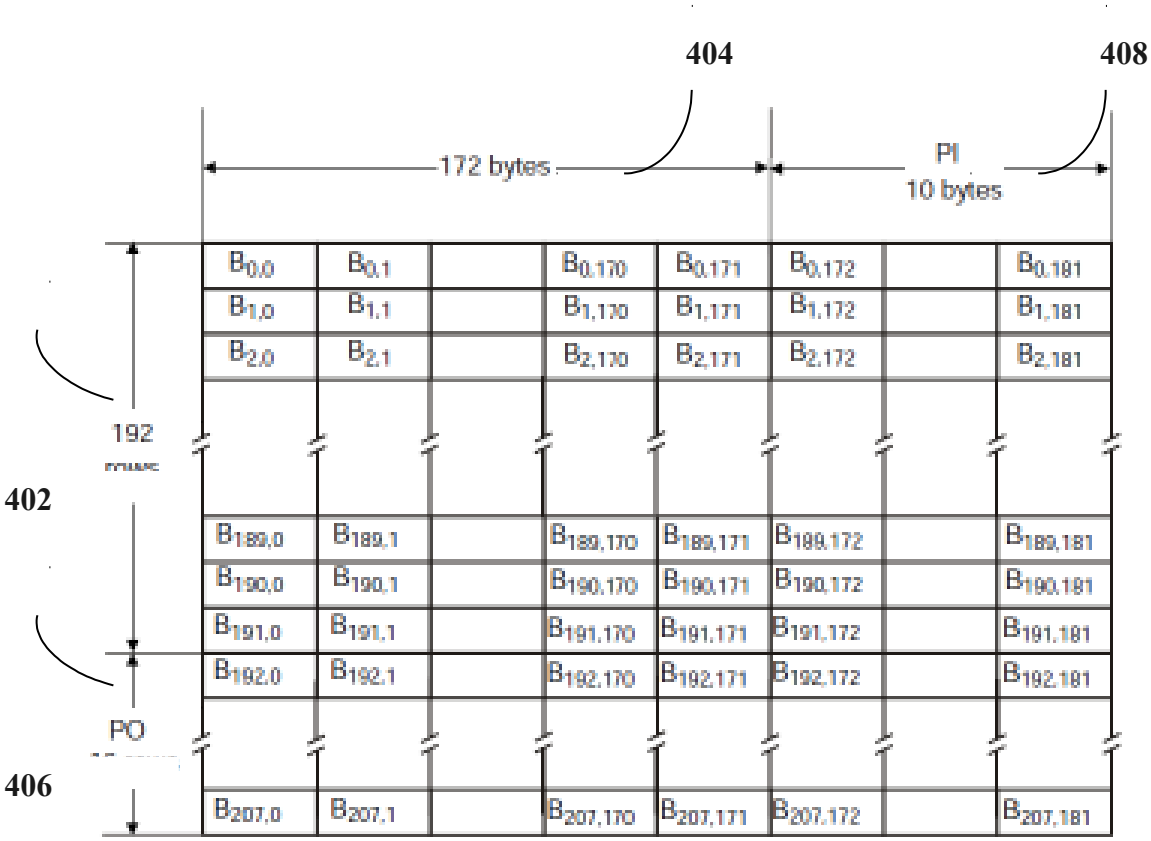

Figure 4

Priyank Gupta

Agent for Applicant
IN-PA-1454

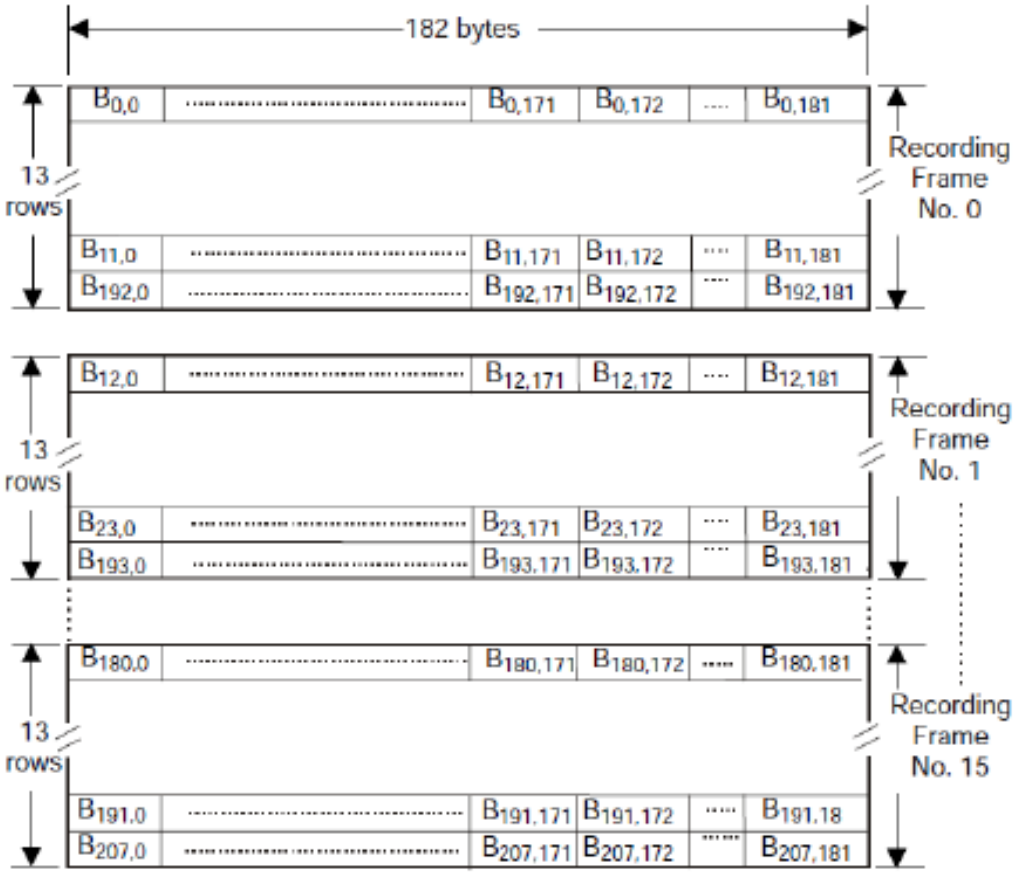

Figure 5

Priyank Gupta

Agent for Applicant
IN-PA-1454