Fusion Coding of Correlated Sources for Storage and Selective Retrieval

Sharadh Ramaswamy, *Member, IEEE*, Jayant Nayak, *Member, IEEE*, and Kenneth Rose, *Fellow, IEEE*

***Abstract*—We focus on a new, potentially important application of source coding directed toward storage and retrieval, termed fusion coding of correlated sources. The task at hand is to efficiently store multiple correlated sources in a database so that, at any point of time in the future, data from a *selective subset* of sources specified by user can be efficiently retrieved. Only statistical information about future *queries* is available in advance. A typical application scenario would be in storage of correlated data generated by dense sensor networks, where information from specific regions is requested in the future. We propose a *fusion coder* (FC) for lossy storage and retrieval, wherein different queries are handled by allowing for selective (compressed) bit retrieval. We derive the properties of an optimal FC and present an iterative algorithm for its design. Since iterative design is initialization-dependent, we present initialization heuristics that help avoid poor local optima. An analysis of design complexity reveals complexity growth with query-set size. We first tackle this problem by exploiting optimality properties of FCs. We also consider quantization of the query-space with decision trees in order to adapt to new queries, unseen during FC design. Experiments conducted on real and synthetic data-sets demonstrate that the proposed FC is able to achieve significantly better tradeoffs than joint compression by vector quantization (VQ), with retrieval speedups reaching 3× and distortion gains of up to 3.5 dB possible.**

***Index Terms*—Database query processing, multisensor systems, source coding, vector quantization (VQ).**

I. Introduction

THIS paper considers the problem of storing correlated sources in a database for future retrieval of any subset of them as queried by users. This problem differs from the well-known distributed source coding setting [1], [2] in that all information about the sources is centrally available during encoding for storage in the database. However, only statistical information about future queries is available. Such database design introduces fundamentally new and interesting challenges. On the one hand, intersource correlations may be exploited via joint coding to reduce the overall storage requirement and to potentially reduce the retrieval time. On the other hand, a future query may select only a few of the sources for retrieval, and it would be wasteful to have to retrieve the entire (jointly) compressed data only to reconstruct a small subset.

Thus, at the heart of the problem lies a tradeoff between storage rate (space) versus retrieval rate (time), both measured in terms of bits stored or retrieved. An example application of the proposed fusion coding of correlated sources is in the arena of sensor networks, which has been the focus of extensive research in recent years. Much of the effort in sensor network design has been dedicated to the development of device and communication technologies [3]. However, in order to fully realize the potential of most such systems, it is necessary to efficiently store the vast volumes of data generated by the network for future retrieval, as needed for analysis or other uses.

Consider a network of surveillance cameras covering a scene. The video signals generated by these cameras are expected to be highly correlated since they are covering the same scene. This data is sent to a fusion center to be stored for possible future analysis. We note in passing the more generalized setting of *multiple* storage centers, each of which store video data from one or more cameras, i.e., *distributed storage*, but for the sake of simplicity, we assume herein that all video signals are stored in a single fusion center. When the data from the fusion center is eventually accessed by a user, it is very likely that the views from only a small subset, and not all, of the cameras will be requested at any given time. Note also that fusion storage of correlated sources has applications even in areas that are far removed from traditional signal processing and communications, such as storage and indexing of stock market data streams [4].

Fig. 1 depicts the setting we consider. The fusion coding problem was first posed by us in [5], where an information-theoretic characterization was derived for the achievable lossless coding rate region via reformulation as a multiterminal source coding problem [6]. By allowing for some loss in quality, the database designer has greater freedom in optimizing tradeoffs between storage rate and retrieval rate. In our early work [7], we derived a lossy fusion coder design scheme that directly optimizes the distortion-retrieval rate tradeoff for memoryless sources. The fusion coder is composed of three modules: encoder, decoder and bit-selector (see Fig. 2). This paper subsumes our preliminary conference publication [7] and, besides a comprehensive treatment of the proposed paradigm and approaches, expands to encompass important practical considerations including reduced complexity variants and methods to avoid poor local optima of the cost surface and enable calculation of operational retrieval rate-distortion curve. We also study methods to quantize the query space in order to handle the large

Manuscript received January 23, 2009; accepted October 29, 2009. First published December 01, 2009; current version published February 10, 2010. The associate editor coordinating the review of this manuscript and approving it for publication was Prof. Pramod K. Varshney. This work was supported in part by the NSF under grants IIS-0329267 and CCF-0728986, the University of California MICRO program, Applied Signal Technology Inc., CISCO Systems Inc., Dolby Laboratories Inc., Qualcomm Inc., and Sony Ericcson Inc.

S. Ramaswamy and J. Nayak are with the Mayachitra Inc., Santa Barbara, CA 93111 USA (e-mail: rsharadh@mayachitra.com).

K. Rose is with the Department of Electrical and Computer Engineering, University of California, Santa Barbara, CA 93106 USA.

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TSP.2009.2037664

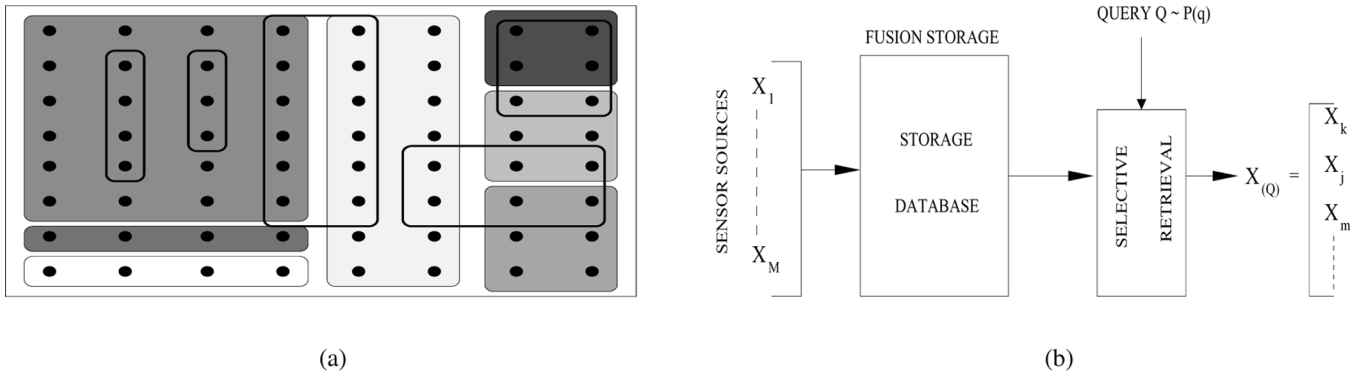

Fig. 1. Fusion coding of correlated sources. (a) A 2D sensor field. Dots represent sensors and boxes represent regions of interest (queries). (b) Fusion storage and selective retrieval.

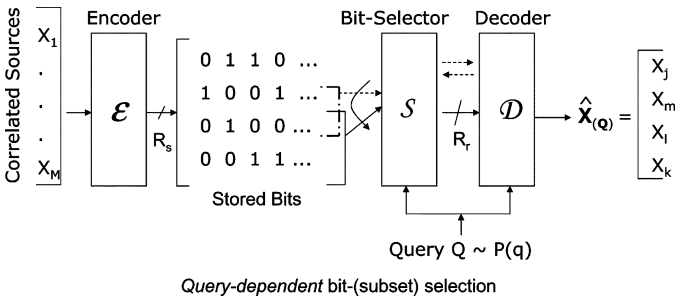

Fig. 2. Proposed fusion coder.

query sets that are typical and adapt to future queries unseen during training.

II. Preliminary Motivation

Let us denote the M correlated sources as the set $\{X_m, m = 1, \ldots, M\}$. We define a query as the subset of sources that need to be retrieved. Employing binary variables $q_i \in \{0, 1\}$ to denote whether source X_i is requested or not, we represent queries by M-tuples of the form

$$\mathbf{q} = (q_1, \ldots, q_M) \in \mathcal{Q} \tag{1}$$

where $\mathcal{Q} \subseteq \{0, 1\}^M$ is the domain-set of queries. We next introduce notation for the query distribution, or the probability mass function (pmf)

$$P : \mathcal{Q} \to [0, 1]. \tag{2}$$

There are 2^M possible queries, and $\sum_{\mathbf{q}\in\mathcal{Q}} P(\mathbf{q}) = 1$. Without loss of generality, we assume that each source is requested with positive probability (i.e., there exists some query with positive probability whose requested subset includes the source) and that a query always asks for a nonempty subset of sources, i.e.,

$$P(\mathbf{0}) = 0. \tag{3}$$

In our notation, boldface letters in lowercase and uppercase represent vectors and random vectors, respectively.

We term each M-tuple entry as a sample. Given a database of constant size (constant number of samples), the *retrieval time* or the time required to retrieve a subset of sources is proportional to the number of bits retrieved *per sample*, which we term the *retrieval rate*. If the number of bits retrieved per sample to answer query $\mathbf{q}$ is $R_{\mathbf{q}}$, then the average retrieval rate is

$$R_r = \sum_{\mathbf{q}\in\mathcal{Q}} P(\mathbf{q}) R_{\mathbf{q}}. \tag{4}$$

Our goal is to *minimize the retrieval time* or equivalently to *minimize the retrieval rate*.

A. Why Compress for Storage?

The rapid development of low-cost, high-density storage devices motivates one to consider why compressed storage is even considered. We note that we are considering storage of data from many correlated sources, the number of which could run from hundreds to several thousands. Hence, the volume of data encountered could overwhelm storage systems, unless compression is performed. A significantly more important issue is that stored data are often going to be requested by users, and without (joint) compression of the requested sources, unnecessarily additional bits would need to be retrieved. Since retrieval time is proportional to the number of retrieved bits, without compression, retrieval times would be very large. Lastly, compression would also be intricately linked with any high-dimensional index for nearest-neighbor search over such databases [8]. For example, the popular VA-File [9] and VA-Stream [4] approaches to similar pattern retrieval use compressed versions of the database to prune the search space and speed up search times.

B. Information Theoretic Bounds on Lossless Storage and Retrieval

To compress any source of information X, the number of bits required for lossless representation should be at least the Shannon entropy (*entropy-rate*, for sources with memory) of the source, $H(X)$. Information theory offers straightforward bounds on the performance of naive compression techniques when applied to the problem of fusion coding.

1) Minimal Storage Rate: It follows from Shannon's basic result that the minimal number of bits required to store M sources $X_1, \ldots, X_M$, i.e., the minimal *storage rate* is

$$R_{s,\min} = H(X_1, \ldots, X_M). \tag{5}$$

Since $H(X_1, \ldots, X_M) \leq \sum_m H(X_m)$, joint compression of correlated sources *never* requires more bits (and usually requires less bits) for storage than separate compression (due to its ability to exploit intersource redundancies). The retrieval rate for this method is similarly

$$R_r = H(X_1, \ldots, X_M) \tag{6}$$

since *the entire compressed description needs to be retrieved for any query*.

2) Minimal Retrieval Rate: If we denote the set of sources queried as

$$X_{(\mathbf{q})} = \{X_m, \forall\, m : q_m = 1\}$$

the minimum number of bits required to reconstruct the sources requested by query $\mathbf{q}$ is $H(X_{(\mathbf{q})})$, and hence, the minimum retrieval rate averaged over the query distribution is

$$R_{r,\min} = \sum_{\mathbf{q}} P(\mathbf{q}) H(X_{(\mathbf{q})}) \leq H(X_1, \ldots, X_M).$$

This implies that joint compression is suboptimal in retrieval speed. In order to achieve the fastest retrieval speed, we need to *separately compress and store each subset* of sources that may be requested.

It is important to note that we are actually faced with a storage and retrieval problem, where the sources are provided to the database designer for storage and *future* retrieval. Individual sources are available during encoding time, but when queries arrive, all we have is the stored data. Minimum retrieval rate requires that the optimal compressed version for a specific query be ready for retrieval. Any reencoding operation would require first to retrieve the necessary information and would hence beat the purpose of minimum retrieval rate. Therefore, encoding on the fly, depending on the query, is not really an option.

On the other hand, unless M is very small or the set of queries $\mathcal{Q}$ is severely restricted, the storage requirement would be impractically high as it would have to individually accommodate a very large (possibly an exponential) number of queries, i.e.,

$$R_s = \sum_{\mathbf{q} \in \mathcal{Q}} H(X_{(\mathbf{q})}) \gg H(X_1, \ldots, X_M) = R_{s,\min}. \tag{7}$$

Thus, it is clear that the optimal storage technique is wasteful in retrieval speed, and the optimal retrieval technique is wasteful in storage.

3) Separate Compression: Last, we consider separate coding of each source. The storage rate for lossless separate compression is

$$R_{s,\text{sep}} = \sum_m H(X_m) \gg R_{s,\min} \tag{8}$$

while the retrieval rate

$$R_{r,\text{sep}} = \sum_{\mathbf{q}} P(\mathbf{q}) \sum_m q_m H(X_m) \gg R_{r,\min}. \tag{9}$$

Clearly, separate coding/quantization would be severely suboptimal both in storage efficiency and retrieval speed. It ignores the correlations that exist across sources, thereby incurring larger storage costs and higher retrieval time than necessary.

III. Fusion Coder Formulation

Consider M real-valued sources $X_m, \forall\, 1 \leq m \leq M$. Any practical signal storage scheme would need to quantize the data before storage, which entails some error or *distortion*. Given query $\mathbf{q}$, the *reconstruction* distortion is measured as

$$d_{\mathbf{q}}(\mathbf{x}, \hat{\mathbf{x}}) = \sum_{m=1}^{M} q_m d_m(x_m, \hat{x}_m) \tag{10}$$

where

$$d_m : \mathbb{R} \times \mathbb{R} \to [0, \infty). \tag{11}$$

Hereafter, we will specialize to the *squared error distortion* measure

$$d_m(x, \hat{x}) = (x - \hat{x})^2, \quad m = 1, \ldots, M. \tag{12}$$

We propose a *fusion coding* framework to optimize the fusion storage-selective retrieval of correlated sources. A block diagram for the fusion coder (FC) is given in Fig. 2. The FC is composed of three modules: encoder, bit (subset)-selector, and decoder. We define the encoder by the function

$$\mathcal{E} : \mathbb{R}^M \to \mathcal{I} = \{0,1\}^{R_s} \tag{13}$$

which compresses the M-dimensional input vector $\mathbf{x}$, representing the M sources, to R_s bits at each instant.

The bit (subset) selector is the mapping

$$\mathcal{S} : \mathcal{Q} \to \mathcal{B} = 2^{\{1, \ldots, R_s\}} \tag{14}$$

where $\mathcal{Q} \subseteq \{0,1\}^M$ represents the domain-set of queries and $\mathcal{B}$ is the power set (set of all subsets) of the set $\{1, \ldots, R_s\}$. This mapping determines which of the stored bits to retrieve for a given query $\mathbf{q}$. Clearly, $\mathcal{S}(\mathbf{q}) \subseteq \{1, \ldots, R_s\}, \forall\, \mathbf{q}$. For each subset of bits $\mathbf{e}$ that can be retrieved, an estimate of all the sources is formed by the decoder

$$\mathcal{D} : \mathcal{I} \times \mathcal{B} \to \hat{\mathcal{X}} \tag{15}$$

where $\hat{\mathcal{X}} \subset \mathbb{R}^M$ is the corresponding codebook. We introduce this notation for the decoder in order to simplify notation for representation of subsets of bits of the encoding index. The decoder does *not* access all the encoded bits, even though one argument of the decoder function is $\mathbf{i}$, where $\mathbf{i}$ is some encoding index. Nor does it assume more information about the sources than allowed by the bit-selector setting for each query.

The average distortion for a specific query $\mathbf{q}$ is

$$D_{\mathbf{q}} = E[d_{\mathbf{q}}(\mathbf{X}, \mathcal{D}(\mathcal{E}(\mathbf{X}), \mathcal{S}(\mathbf{q})))] \tag{16}$$

where $E[\ldots]$ denotes statistical expectation, and the distortion averaged across all queries is $D = \sum_{\mathbf{q}\in\mathcal{Q}} P(\mathbf{q})D_{\mathbf{q}}$. In practice, we only have access to the database $\mathcal{X}$ and not necessarily the underlying joint probability distribution function. We assume that the database samples are drawn i.i.d from an unknown joint distribution and use the database as a training set. We then replace the expectation operator $E[\ldots]$ by a simple average, evaluated across the database $\mathcal{X}$. This practice is justified by the ergodic property that with a large enough sample set, the sample average approaches the expected value. We note that this is a standard procedure (in compression and statistical pattern recognition) in the use of a training set to estimate expectation over underlying unknown joint distribution. Hence, the distortion is evaluated as

$$D = \sum_{\mathbf{q}\in\mathcal{Q}} P(\mathbf{q}) \frac{1}{|\mathcal{X}|} \sum_{\mathbf{x}\in\mathcal{X}} d_{\mathbf{q}}(\mathbf{x}, \hat{\mathbf{x}}). \tag{17}$$

Noting that $R_{\mathbf{q}} = R_{\mathcal{S}(\mathbf{q})} = |\mathcal{S}(\mathbf{q})|$, the average retrieval rate is

$$R_r = \sum_{\mathbf{q}} P(\mathbf{q}) R_{\mathbf{q}} = \sum_{\mathbf{q}} P(\mathbf{q}) |\mathcal{S}(\mathbf{q})|. \tag{18}$$

Given M correlated sources, the designer aims to minimize distortion (or maximize quality) and at the same time minimize retrieval rate and storage rate, which necessitates a tradeoff between the three competing quantities. In our approach, we fix the storage rate and optimize the retrieval rate-distortion tradeoff. This choice is not fundamentally necessary, but is motivated by practical system design, where the designer typically knows the storage capacity (RAM/hard-disk capacity), or effectively the storage rate, and where the tradeoff between distortion and retrieval rate is the remaining design flexibility. This is equivalent to minimizing the corresponding Lagrangian

$$\min_{\mathcal{E},\mathcal{S},\mathcal{D}} J = \min_{\mathcal{E},\mathcal{S},\mathcal{D}} D(R_s) + \lambda R_r(R_s), \quad \lambda \geq 0 \tag{19}$$

where the Lagrange multiplier λ controls the tradeoff.

We note that this Lagangian formulation is well known in optimization literature [10] and is equivalent to minimizing the distortion subject to a retrieval rate constraint, where varying the value of the λ parameter provides optimal solutions at varying levels of the retrieval rate constraint. Indeed, the Lagrangian formulation is basically tradeoff control between competing quantities, and the solution is easily converted to different options of which quantity is constrained and which is optimized. A classical example of such easy conversion are the rate-distortion and distortion-rate functions of information theory (see [11]).

IV. Necessary Conditions for Optimality

The Lagrangian cost J can be rewritten as

$$J = \frac{1}{|\mathcal{X}|} \sum_{\mathbf{x}} \sum_{\mathbf{q}} P(\mathbf{q}) d_{\mathbf{q}}(\mathbf{x}, \mathcal{D}(\mathcal{E}(\mathbf{x}), \mathcal{S}(\mathbf{q}))) + \lambda \sum_{\mathbf{q}} P(\mathbf{q}) |\mathcal{S}(\mathbf{q})|. \tag{20}$$

1) Optimal Encoder: From equation (20), the optimal encoding index $\mathcal{E}(\mathbf{x})$ for input vector $\mathbf{x}$ satisfies

$$\mathcal{E}(\mathbf{x}) = \arg\min_{\mathbf{i}\in\mathcal{I}} \sum_{\mathbf{q}} P(\mathbf{q}) d_{\mathbf{q}}(\mathbf{x}, \mathcal{D}(\mathbf{i}, \mathcal{S}(\mathbf{q}))), \quad \forall\, \mathbf{x}. \tag{21}$$

It is easy to see the resemblance to the "nearest neighbor" condition in quantizer design [12].

2) Optimal Bit-Selector: Similarly, the best set of bits to retrieve for a particular query should be the one that minimizes the query's contribution to the Lagrangian cost of (20)

$$\mathcal{S}(\mathbf{q}) = \arg\min_{\mathbf{e}\in\mathcal{B}} \Big\{ \frac{1}{|\mathcal{X}|} \sum_{\mathbf{x}} d_{\mathbf{q}}(\mathbf{x}, \mathcal{D}(\mathcal{E}(\mathbf{x}), \mathbf{e})) + \lambda |\mathbf{e}| \Big\}, \quad \forall\, \mathbf{q}. \tag{22}$$

3) Optimal Decoder: Let $\mathbf{i} \in \mathcal{I}$ be an encoding index and $\mathbf{e} \in \mathcal{B}$ represent some subset of bits. We use $\mathbf{i}_{\mathbf{e}}$ to denote the subindex extracted from $\mathbf{i}$ by retrieving the bits in the positions indicated by $\mathbf{e}$ and $\mathcal{D}(\mathbf{i}, \mathbf{e})$ to denote the corresponding codevector. Clearly, the choice of codevector does not affect the rate component of the Lagrangian cost J.

Let $F_{\mathbf{i},\mathbf{e}} = \{\mathbf{x} : (\mathcal{E}(\mathbf{x}))\mathbf{e} = \mathbf{i}_{\mathbf{e}}\}$ and $E_{\mathbf{e}} = \{\mathbf{q} : (\mathcal{S}(\mathbf{q})) = \mathbf{e}\}$. We can rewrite the distortion as

$$\begin{aligned} D &= \frac{1}{|\mathcal{X}|} \sum_{\mathbf{i},\mathbf{e}} \sum_{m} \sum_{\mathbf{q}\in E_{\mathbf{e}}} \sum_{\mathbf{x}\in F_{\mathbf{i},\mathbf{e}}} P(\mathbf{q}) q_m (x_m - \mathcal{D}(\mathbf{i}, \mathbf{e})_m)^2 \\ &= \frac{1}{|\mathcal{X}|} \sum_{\mathbf{i},\mathbf{e}} \sum_{m} w_m \sum_{\mathbf{x}\in F_{\mathbf{i},\mathbf{e}}} (x_m - \mathcal{D}(\mathbf{i}, \mathbf{e})_m)^2 \end{aligned} \tag{23}$$

where $w_m = \sum_{\mathbf{q}\in E_{\mathbf{e}}} P(\mathbf{q}) q_m$.

In the optimal decoder, the partial derivatives of J with respect to the codevector $\mathcal{D}(\mathbf{i}, \mathbf{e})$, $\forall\, \mathbf{i}, \mathbf{e}$ are 0.

$$\frac{\partial J}{\partial \mathcal{D}(\mathbf{i}, \mathbf{e})} = 0 \Rightarrow \mathcal{D}(\mathbf{i}, \mathbf{e}) = \frac{1}{|F_{\mathbf{i},\mathbf{e}}|} \sum_{\mathbf{x}\in F_{\mathbf{i},\mathbf{e}}} \mathbf{x}, \quad \forall\, \mathbf{i}, \mathbf{e}. \tag{24}$$

We note in passing that this necessary property of the optimal decoder is reminiscent of the "centroid" rule in quantizer design [12].

A. Algorithm for Fusion Coder Design

Since we are considering the storage and retrieval of signals from a database, the signals from all sources are already available. Hence, while one would use the entire database as a training set, it is equally important to note that the database is also the *test set*. A natural design algorithm is to iteratively enforce each of the necessary conditions for optimality (derived in the preceding sections), until a convergence condition is satisfied. In effect, the algorithm just partitions the elements of the training set and the storage bits into different groups, and for a finite-sized training set and a finite storage rate, there exist only a finite number of set partitions. At each step of the optimization, a subset of parameters is adjusted to minimize the Lagrangian cost. Since the cost is nonincreasing at each step, the algorithm is guaranteed to converge in a finite number of iterations. It is to be noted that since the Lagrangian cost surface is nonconvex

and has multiple local optima, iterative design would be initialization dependent and may *not* lead to a *globally optimal* solution. Better (but more complex) optimization techniques such as deterministic annealing [13] may be necessary to approach the global optimum.

V. Design Complexity

This section begins with a discussion of complexity of design and operation. We later exploit the optimality properties of the FC and provide heuristics to accelerate the design procedure.

At the encoder, the search for the optimal encoding index for each element $\mathbf{x}$ in the training set/database $\mathcal{X}$ involves 2^{R_s} distance evaluations of the form $\sum_{\mathbf{q}} P(\mathbf{q}) d_{\mathbf{q}}(\mathbf{x}, \mathcal{D}(\mathbf{i}, \mathcal{S}(\mathbf{q})))$. This implies a net complexity cost $O(2^{R_s}|\mathcal{X}||\mathcal{Q}|M)$ additions and multiplications in encoder optimization. Codebook optimization, on the other hand, involves computation of averages of different subsets of the training set, and this complexity grows as $O(|\mathcal{X}|)$ for each codevector. Additionally, the total number of codevectors that need to be maintained is $\sum_{k=1}^{R_s} \binom{R_s}{k} 2^k = 3^{R_s} - 1$. Each vector average is over M components, and hence codebook update involves $O(3^{R_s}|\mathcal{X}|M)$ addition operations.

The optimization of the bit(-subset) selector is effectively a search for the best subset among the set of $2^{R_s} - 1$ possible subsets for each query $\mathbf{q}$. This involves computation of the average distortion across the entire data-set, over all possible bit-subsets for each query. This implies that the complexity of this step grows as $O(2^{R_s}|\mathcal{X}||\mathcal{Q}|M)$. We also note that the storage complexity of the bit-selector, which is just a lookup table, grows as $O(|\mathcal{Q}|)$. Clearly, the complexity of FC design scales linearly with the size of the database, the query set, and the number of sources, but exponentially with R_s.

A. Complexity of Operation

Once the system has been designed, the encoding of the database has been completed. Hence, during usage/deployment, only the decoding is performed. For each query, the optimal subsets of encoded bits are retrieved, and the relevant sources are reconstructed. However, if only a small part of the database was used to train the system, either to speed up training or because these entries were unavailable during the FC design phase, any remaining/new entries would of course need to be encoded (entailing the corresponding encoder usage complexity).

B. Complexity Reduction Strategies

We note that the query-set $\mathcal{Q}$ could be very large as could be the number of sources considered and the database itself. A first attempt at reducing complexity would be to limit the training set to be a statistically representative subset and not the entire database. However, the design complexity may still be unacceptable as it is a product of the sizes of the training set, query set, and the number of sources. In this section, we exploit properties of the optimal FC to further reduce the design complexity.

1) Faster Encoding With an Average Codebook: At first glance, the assignment of the optimal encoding index to each input vector $\mathbf{x}$ [see (21)] involves $O(|\mathcal{Q}|2^{R_s})$ operations. However, properties of the squared-error distortion measure may be exploited to reduce encoding complexity. Noting that $d_{\mathbf{q}}(\mathbf{x}, \mathbf{0}) = \sum_m q_m x_m^2$, it is easy to see

$$\begin{aligned} d_{\mathbf{q}}(\mathbf{x}, \mathbf{y}) &= d_{\mathbf{q}}(\mathbf{x}, \mathbf{0}) - 2\sum_m q_m x_m y_m + d_{\mathbf{q}}(\mathbf{y}, \mathbf{0}) \\ &= d_{\mathbf{q}}(\mathbf{x}, \mathbf{0}) - 2\mathbf{x}^T W_{\mathbf{q}} \mathbf{y} + d_{\mathbf{q}}(\mathbf{y}, \mathbf{0}) \end{aligned}$$

where the matrix $W_{\mathbf{q}}$ is diagonal, with $W_{\mathbf{q},m,m} = q_m$ This implies that

$$\begin{aligned} &\sum_{\mathbf{q}} P(\mathbf{q}) d_{\mathbf{q}}(\mathbf{x}, \mathcal{D}(\mathbf{i}, \mathcal{S}(\mathbf{q}))) \\ &\quad = \sum_{\mathbf{q}} P(\mathbf{q}) \Big\{ d_{\mathbf{q}}(\mathbf{x}, \mathbf{0}) - 2\mathbf{x}^T W_{\mathbf{q}} \mathcal{D}(\mathbf{i}, \mathcal{S}(\mathbf{q})) \\ &\qquad\qquad + d_{\mathbf{q}}(\mathcal{D}(\mathbf{i}, \mathcal{S}(\mathbf{q})), \mathbf{0}) \Big\} \\ &\quad = \sum_{\mathbf{q}} P(\mathbf{q}) d_{\mathbf{q}}(\mathbf{x}, \mathbf{0}) - 2\bar{\mathbf{c}}(\mathbf{i})^T \mathbf{x} + \alpha_{\mathbf{i}}. \end{aligned}$$

The first term is common to all encoding indices and hence need not be computed. The second term could be viewed as an inner (dot) product of $\mathbf{x}$ with a *query-averaged* codevector $\bar{\mathbf{c}}(\mathbf{i}) = \sum_{\mathbf{q}} P(\mathbf{q}) W \mathbf{q} \mathcal{D}(\mathbf{i}, \mathcal{S}(\mathbf{q}))$, and the last term, $\alpha_{\mathbf{i}}$, is a constant independent of $\mathbf{x}$. $\alpha_{\mathbf{i}}$ and $\bar{\mathbf{c}}(\mathbf{i})$ can be computed in a separate step prior to the encoding of all $\mathbf{x}$. Thus, we obtain the faster encoding rule

$$\mathcal{E}(\mathbf{x}) = \arg\min_{\mathbf{i} \in \mathcal{I}} \alpha_{\mathbf{i}} - 2\bar{\mathbf{c}}(\mathbf{i})^T \mathbf{x}, \quad \forall\, \mathbf{x}. \tag{25}$$

The new encoder design/operation is now of $O(2^{R_s}|\mathcal{X}|M + 2^{R_s}|\mathcal{Q}|M) \approx O(2^{R_s}|\mathcal{X}|M)$ complexity since the computation of the average codebook is a one-time affair prior to encoding the entire database (training set).

2) Recursive Decoder Codevector Update: Given R_s storage bits, there are $2^{R_s} - 1$ codebooks, one for each nonempty subset of bits. We denote the codevector based on all R_s encoded bits for index $\mathbf{j}$ as $\hat{\mathbf{x}}_{\mathbf{j}} = \mathcal{D}(\mathbf{j}, \{1, \ldots, R_s\})$.

Let $F_{\mathbf{i},\mathbf{e}} = \{\mathbf{x} : (\mathcal{E}(\mathbf{x}))\mathbf{e} = (\mathbf{i})_{\mathbf{e}}\}$, $G_{\mathbf{j}} = \{\mathbf{x} : \mathcal{E}(\mathbf{x}) = \mathbf{j}\}$, and $H_{\mathbf{i},\mathbf{e}} = \{\mathbf{j} \in \mathcal{I} : \mathbf{j}_{\mathbf{e}} = \mathbf{i}_{\mathbf{e}}\}$. It is easy to see that

$$F_{\mathbf{i},\mathbf{e}} = \bigcup_{\mathbf{j} \in H_{\mathbf{i},\mathbf{e}}} G_{\mathbf{j}} \tag{26}$$

and that $G_{\mathbf{j}} \bigcap G_{\mathbf{k}} = \phi, \forall\, \mathbf{j} \neq \mathbf{k}$. Consequently

$$\sum_{\mathbf{x} \in F_{\mathbf{i},\mathbf{e}}} \mathbf{x} = \sum_{\mathbf{j} \in H_{\mathbf{i},\mathbf{e}}} \sum_{\mathbf{x} \in G_{\mathbf{j}}} \mathbf{x} \tag{27}$$

$$|F_{\mathbf{i},\mathbf{e}}| = \sum_{\mathbf{j} \in H_{\mathbf{i},\mathbf{e}}} |G_{\mathbf{j}}|. \tag{28}$$

However, (24) also implies

$$\hat{\mathbf{x}}_{\mathbf{j}} = \frac{1}{|G_{\mathbf{j}}|} \sum_{\mathbf{x} \in G_{\mathbf{j}}} \mathbf{x}. \tag{29}$$

Hence

$$\mathcal{D}(\mathbf{i}, \mathbf{e}) = \frac{1}{|F_{\mathbf{i},\mathbf{e}}|} \sum_{\mathbf{x} \in F_{\mathbf{i},\mathbf{e}}} \mathbf{x} = \frac{1}{|F_{\mathbf{i},\mathbf{e}}|} \sum_{\mathbf{j} \in H_{\mathbf{i},\mathbf{e}}} |G_{\mathbf{j}}| \hat{\mathbf{x}}_{\mathbf{j}}.$$

In other words, codevector update is effectively the *weighted (vector) average* of the corresponding codevectors obtained by extracting all the stored bits. Hence, codebook update can be performed recursively starting off from the updates of $\mathcal{D}(\mathbf{j}, \{1, \ldots, R_s\})$, $\forall\ \mathbf{j}$, and is now of $O(2^{R_s}|\mathcal{X}|M + (3^{R_s} - 2^{R_s})M) \approx O(2^{R_s}|\mathcal{X}|M)$ complexity. Note that the storage complexity is reduced from $O(3^{R_s})$ to $O(2^{R_s})$ codevectors. We store $\hat{\mathbf{x}}_{\mathbf{j}}$ and $|G_{\mathbf{j}}|$, $\forall\ \mathbf{j}$, and extract all other codevectors by appropriate averaging.

3) Reduced Complexity Bit-Selector Optimization: The optimal bit-selector satisfies

$$\mathcal{S}(\mathbf{q}) = \arg\min_{\mathbf{e}\in\mathcal{B}} \left\{ \frac{1}{|\mathcal{X}|} \sum_{\mathbf{x}} d_{\mathbf{q}}(\mathbf{x}, \mathcal{D}(\mathcal{E}(\mathbf{x}), \mathbf{e})) + \lambda|\mathbf{e}| \right\}, \quad \forall\, \mathbf{q}.$$

Let $D_m(\mathbf{e}) = 1/|\mathcal{X}| \sum_{\mathbf{x}} (x_m - \mathcal{D}(\mathcal{E}(\mathbf{x}), \mathbf{e})_m)^2$, $\forall\ m$, $\forall\ \mathbf{e} \subseteq \{1, \ldots, M\}$. Now, by interchanging the order of summation

$$\mathcal{S}(\mathbf{q}) = \arg\min_{\mathbf{e}\in\mathcal{B}} \sum_{m} q_m D_m(\mathbf{e}) + \lambda|\mathbf{e}|, \quad \forall\, \mathbf{q} \tag{30}$$

which implies that we could compute $D_m(\mathbf{e})$, $\forall\ m$, $\forall\ \mathbf{e}$, in a separate step and use this result in finding the optimal $\mathcal{S}(\mathbf{q})$, $\forall\ \mathbf{q}$. Hence, the complexity of bit-selector optimization reduces to $O(2^{R_s}|\mathcal{X}|M + |\mathcal{Q}|M) \approx O(2^{R_s}|\mathcal{X}|M)$.

VI. Initialization Strategies

We note that there are two mappings that require initialization—the codebook $\hat{\mathcal{X}}$ and the bit-selector $\mathcal{S}$. As described in Section IV-A, FC design is iterative and dependent on initialization. Additionally, the cost surface is nonconvex and riddled with local minima. Therefore, for any given λ, FC design should be performed with different (possibly random) initializations in order to avoid poor local minima. In this section, we describe some initialization heuristics that would help avoid some poor minima. (Global optimization techniques may be used, but are beyond the scope of this paper.)

A. Computation of Operational Retrieval Rate-Distortion Curve

Suppose for some λ, a good codebook and bit-selector are known. This can be a good initialization point for other (R_r, D) points. For an incrementally different value of λ, we start off with the same codebook, iteratively optimize the bit-selector, the encoder, and decoder (in that order) till convergence. Alternatively, we could retain the same bit-selection and iteratively optimize the encoder, decoder, and bit-selector (in that order). This process could be used to gradually compute the entire rate distortion curve.

Now, for an arbitrary λ, it is unclear how best to initialize the bit-selection and codebooks. However, there are two extreme settings where good heuristics for bit-selector initialization are possible. By starting at either of these points, it is possible to sweep through the entire curve. This would alleviate initialization issues to some extent, even though multiple runs with random codebook initialization could still be necessary to obtain good results.

1) Retrieve All Bits or $\lambda = 0$: We first consider the special case when $\lambda = 0$. This implies Lagrangian cost is solely composed of distortion and that the penalty for bit retrieval is zero. In other words, the optimal bit-selector setting is to *retrieve all compressed bits*. This initialization setting would work for *all* query sets.

2) Retrieve Only One Bit or $\lambda = \infty$: Next, we consider the case when $\lambda = \infty$ (or in practice, a very large value). This implies that the Lagrangian cost is dominated by R_r, while distortion plays almost no role. Since the FC is constrained to retrieve at least one bit, the bit-selector setting must do exactly that, i.e., retrieve exactly one bit for any query. Since there are R_s encoding bits, by retrieving one bit per query, the query set is partitioned into groups of queries, where the same bit is retrieved by the bit-selector. We note that when $|\mathcal{Q}| \leq R_s$, each group consists of only one query, and some storage bits could remain unused. When $|\mathcal{Q}| \geq R_s$, we ensure that R_s nonempty groups are created. This partitioning of the query set is necessary so that all allowed R_s bits (all degrees of freedom) are used during encoding. We note that such partitioning of the query set would clearly be possible in some query distributions, such as those with multiple modes. In other settings, this partitioning may not be very clear, and the alternate initialization scheme (Section VI-A1) would be preferable.

VII. Adapting to Large/Incomplete Query Training Sets

In the fusion coder, we note that the bit-selector is a lookup table that grows with the size of the query-set. In principle, the query-set might be very large. For example, if $M = 100$ sources and suppose any query of size 20 can be requested, then $|\mathcal{Q}| = \binom{100}{20} \approx 10^{20}$, which would impose an unbearable storage requirement. In an extreme case, the query-set may be the entire distribution, i.e., $|\mathcal{Q}| = 2^M - 1$. Even otherwise, the query-set may change after training, i.e., a different set of queries (nevertheless drawn from the same distribution) might be encountered during operation. In either case, the queries need to be classified (grouped), where all queries in a group share the same bit-selection (combination of bits).

Let the allowed number of groupings be L. We define the query-classifier as the mapping

$$\mathcal{C} : \mathcal{Q} \to \mathcal{L} = \{1, \ldots, L\}. \tag{31}$$

$\mathcal{C}$ defines L disjoint partitions of the query-space $\{B_l\}_{l=1}^{L}$ such that

$$B_l = \{\mathbf{q} : \mathcal{C}(\mathbf{q}) = l\}, \quad \forall\, l = 1, \ldots, L \tag{32}$$

$$\Rightarrow \bigcup_{l=1}^{L} B_l = \mathcal{Q}, \quad B_l \bigcap B_m = \phi, \quad \forall\, l \neq m. \tag{33}$$

The bit-selector is modified to be the mapping

$$\mathcal{S} : \mathcal{L} \to \mathcal{B} = 2^{\{1,\ldots,R_s\}} \tag{34}$$

where the notations have the usual meaning. The Lagrangian cost to be optimized is

$$J = \sum_{l} \sum_{\mathbf{q} \in B_l} P(\mathbf{q}) \times \left\{ \frac{1}{|\mathcal{X}|} \sum_{\mathbf{x}} d_{\mathbf{q}}(\mathbf{x}, \mathcal{D}(\mathcal{E}(\mathbf{x}), \mathcal{S}(l))) + \lambda |\mathcal{S}(l)| \right\}. \quad (35)$$

Now, $\mathcal{C}$ must be optimized based on the available (training) set of queries and should be capable of accurately classifying an unseen test query. This might require some structural constraint to be imposed on $\mathcal{C}$, such as the nearest neighbor classifier, nearest prototype classifier, decision tree, etc. This structure can be either adjusted during the optimization of all mappings $\mathcal{E}$, $\mathcal{D}$, $\mathcal{S}$ or optimized offline (after which $\mathcal{E}$, $\mathcal{D}$, $\mathcal{S}$ would need to be reoptimized). In subsequent sections, we shall confine discussion to the latter option.

A. Necessary Conditions for Optimality

We now present the optimality conditions for this fusion coder (avoiding lengthy derivations).

1) Optimal Encoder: From (35), the optimal encoding index $\mathcal{E}(\mathbf{x})$ for input vector $\mathbf{x}$ is

$$\mathcal{E}(\mathbf{x}) = \arg\min_{\mathbf{i} \in \mathcal{I}} \sum_{l=1}^{L} \sum_{\mathbf{q} \in B_l} P(\mathbf{q}) d_{\mathbf{q}}(\mathbf{x}, \mathcal{D}(\mathbf{i}, \mathcal{S}(l))), \quad \forall\, \mathbf{x}. \quad (36)$$

2) Optimal Bit-Selector: Similarly, the best set of bits to retrieve for a particular label (query-region/partition) is the one that minimizes the Lagrangian sum of the distortion measure $\sum_{\mathbf{q} \in B_l} P(\mathbf{q}) d_{\mathbf{q}}(.,.)$, averaged over the training set, and the retrieval rate, i.e.,

$$\mathcal{S}(l) = \arg\min_{\mathbf{e} \in \mathcal{B}} \sum_{\mathbf{q} \in B_l} P(\mathbf{q}) \times \left\{ \frac{1}{|\mathcal{X}|} \sum_{\mathbf{x}} d_{\mathbf{q}}(\mathbf{x}, \mathcal{D}(\mathcal{E}(\mathbf{x}), \mathbf{e})) + \lambda |\mathbf{e}| \right\}, \quad \forall\, l. \quad (37)$$

3) Optimal Query-Classification: The optimal labeling (grouping) of the queries would be

$$\mathcal{C}(\mathbf{q}) = \arg \min_{1 \le l \le L} \frac{1}{|\mathcal{X}|} \sum_{\mathbf{x}} d_{\mathbf{q}}(\mathbf{x}, \mathcal{D}(\mathcal{E}(\mathbf{x}), \mathcal{S}(l))) + \lambda |\mathcal{S}(l)|. \quad (38)$$

4) Optimal Decoder: We use $\mathbf{i}_{\mathbf{e}}$ to denote the subindex extracted from $\mathbf{i}$ by retrieving the bits in the positions indicated by $\mathbf{e}$ and $\mathcal{D}(\mathbf{i}, \mathbf{e})$ to denote the corresponding codevector. By setting to zero, the partial derivatives of J with respect to $\mathcal{D}(\mathbf{i}, \mathbf{e}) \,\forall\, \mathbf{e} \in \mathcal{B}$, we obtain the optimal decoder to be

$$\mathcal{D}(\mathbf{i}, \mathbf{e}) = \frac{1}{|F_{\mathbf{i},\mathbf{e}}|} \sum_{\mathbf{x} \in F_{\mathbf{i},\mathbf{e}}} \mathbf{x}, \quad \forall\, \mathbf{e}, \mathbf{i} \quad (39)$$

where $F_{\mathbf{i},\mathbf{e}} = \{\mathbf{x} : (\mathcal{E}(\mathbf{x}))\mathbf{e} = (\mathbf{i})_{\mathbf{e}}\}$.

B. Algorithm for Design

We design the fusion coder by iteratively applying the conditions for optimality. Upon convergence, we learn the parameters of the structure imposed on $\mathcal{C}$ such as the centroids, prototypes, the nodes to be split etc. Given the structure of $\mathcal{C}$, we reoptimize $\mathcal{E}$, $\mathcal{D}$, $\mathcal{S}$.

VIII. Experimental Results

A. Data-Sets

We tested our algorithm extensively on both synthetic and real data-sets, where we evaluated the operational (retrieval) rate (R_r) versus distortion D curves for different settings of storage complexity. A brief description of our data-sets follows.

1) SYNTH: Synthetic Data: The sensor sources were modeled as zero-mean, correlated Gaussian sources (of unit variance i.e., $\sigma^2 = 1$) independently and identically drawn from a jointly Gaussian density, with the correlation between sources modeled as falling exponentially with distance. Specifically, if ρ_{ij} represents the correlation between sources X_i and X_j

$$\rho_{ij} = \rho^{|i-j|} \quad (40)$$

where $-1 \le \rho \le 1$. This correlation model can be expected when spatio-temporal sensor fields are uniformly sampled [14]. We created synthetic data-sets with $\rho = 0.3$ and $\rho = 0.8$, corresponding to low and moderately high correlation data-sets, with $M = 50$ sources, each having 6000 training samples and 1 000 000 test samples. The performance on the test set is reported.

2) STOCKS: Real Data: The first real data-set, the STOCKS data-set, is available in the University of California, Riverside (UCR) Time-Series Data Mining Archive.[1] It consists of $M = 93$ stocks, each having 3000 samples.

3) Intel Berkeley Sensor Data: The second real data-set used was the one generated by the Intel Berkeley Research Lab, CA.[2] Data were collected from 54 sensors deployed in the Intel Berkeley Research Lab between February 28 and April 5, 2004. Each sensor measures humidity, temperature, light, and voltage values once every 31 s. We retain data from those sensors that generated in excess of 50 000 readings. This corresponds to temperature, light, humidity, and voltage readings from 15 sensors, which is equivalent to 60 sources.

B. Query Distribution

We tested the performance of the fusion coder on several query distributions that model real user behavior. Even though, in theory, there are $2^M - 1$ possible queries, typically only a smaller subset of sources (say n) will normally be requested at any time. There are $\binom{M}{n}$ ways of selecting n out of M objects, and for moderate values of M, even this might be very large. For example, if $M = 30$ and $n = 4$, $2^{30} - 1 \simeq 10^9$ and $\binom{30}{4} = 27\,405$.

[1]The authors would like to thank Dr. E. Keogh of UCR for kindly providing the STOCKS data-set.

[2]Download from http://db.csail.mit.edu/labdata/labdata.html.

Fig. 3. "Neighborhoods" of sources (on a 1-D sensor array) of varying size.

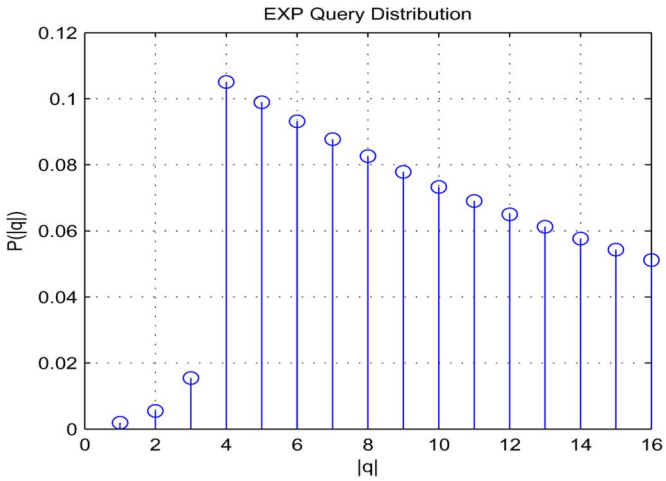

Fig. 4. EXP: Exponential distribution on "neighborhood" (query) sizes for $M = 50$ sources.

We describe *exponential queries* ("EXP"), which we believe are reasonable models for real-user behavior. Queries request for contiguous *neighborhoods* of sources (see Fig. 3). We impose a shifted exponential distribution on neighborhood size. In our sample distribution, on an average, nine are requested, and it is representative of 345 queries, which were randomly generated. Even though the queries were chosen randomly, we ensured that each source is requested by at least one query. Fig. 4 is representative of this query distribution. The probability is plotted versus the *size* of the query $|\mathbf{q}|$. It is also to be noted that even a query set of size 345 *cannot be handled* with the naive storage technique presented in Section II-B2, i.e., compressing and storing every subset of sources separately, without paying an enormous price in total storage.

C. Fusion Coding (FC) versus Joint Compression (VQ)

We compared the performance of joint compression and selective bit-retrieval for both the synthetic and real data-sets (see Figs. 5–7). Joint compression of the data-set was performed by a vector quantizer (VQ) designed with the standard Generalized Lloyd Algorithm (GLA) [12]. We note that in joint compression, all the compressed bits are retrieved, i.e., $R_r = R_s$. Since the designer has a handle only on R_s, different points on the operational retrieval rate-distortion curve are obtained by varying the storage (compression) rate R_s. In experiments on all data-sets, for VQ, $1 \leq R_s \leq 6$ bits. For the proposed FC, performance was evaluated at two storage settings, $R_s = 4$ and $R_s = 6$ bits. Since both systems are designed iteratively, they are initialization-dependent. We performed 20 different runs with random codebook initializations (for both VQ and FC) and present the best performance of each method. This amounts to plotting the convex hull of the various (R_r, D) points obtained in each method.

Note the fundamental shortcoming of VQ, in that it ties storage and retrieval rates together. On the other hand, FC has additional degrees of freedom through which it is able to reduce retrieval rate without changing storage rate, leading to substantial gains over VQ. For the synthetic data-set with $\rho = 0.3$ (Fig. 5), FC is able to provide a speedup of nearly $3\times$ at a distortion level of 9.1 dB. For the synthetic data-set with $\rho = 0.8$ (Fig. 6), there is a $\approx 3\times$ speedup over the joint compression technique, with average distortion of 8 dB. Additionally, there is also a distortion gain of nearly 1 dB at a retrieval rate of 1 bit per sample. Since each source is a unit variance Gaussian, it also has unit energy. On the average, nine sources are retrieved, which implies that the average "signal energy" is 9.5 B. This also equals the distortion when $R_r = 0$ bits per sample, i.e., when no information is retrieved from the database. Thus, compression by FC or VQ leads up to 0.7 dB reduction in distortion for the $\rho = 0.3$ data-set and 2.7 dB reduction in distortion for the $\rho = 0.8$ dataset.

In the real data-set STOCKS (Fig. 7), FC provides a $\approx 1.6\times$ speedup with distortion 28 dB and nearly 3.5 dB less distortion

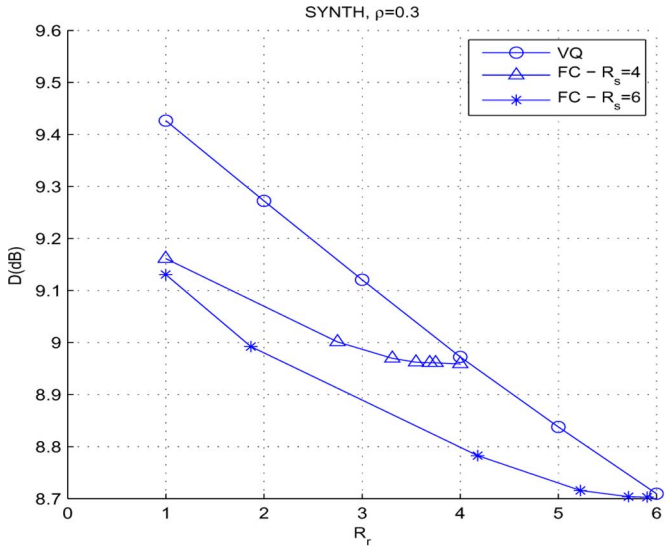

Fig. 5. Data-set SYNTH, $\rho = 0.3$.

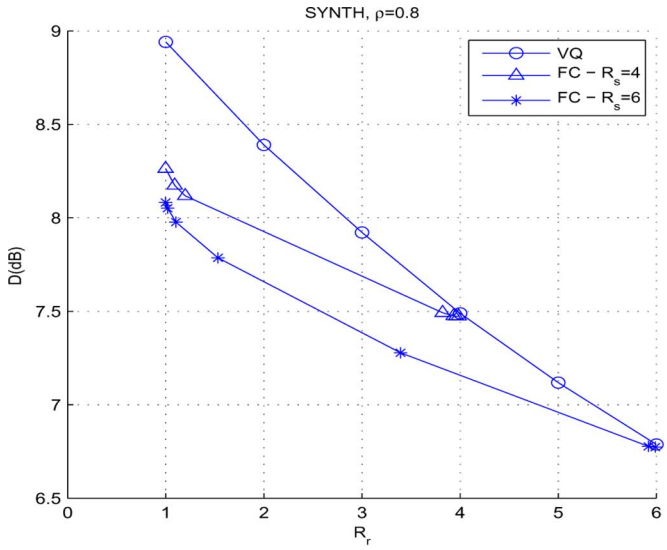

Fig. 6. Data-set SYNTH, $\rho = 0.8$.

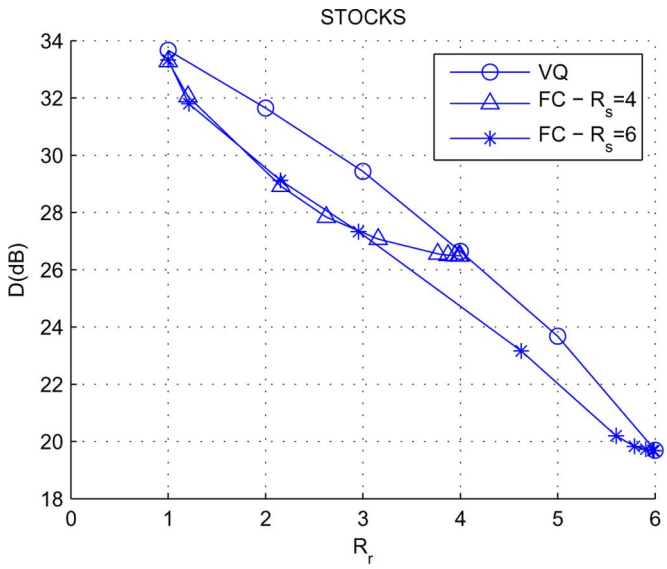

Fig. 7. Data-set STOCKS.

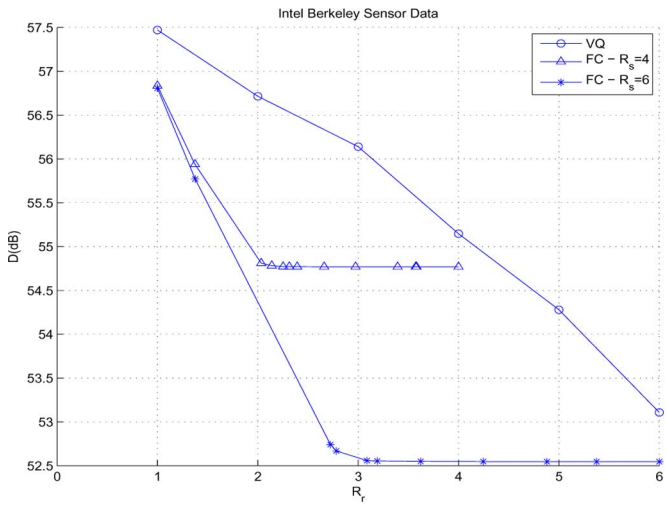

Fig. 8. Data-set Intel Berkeley Sensor Data.

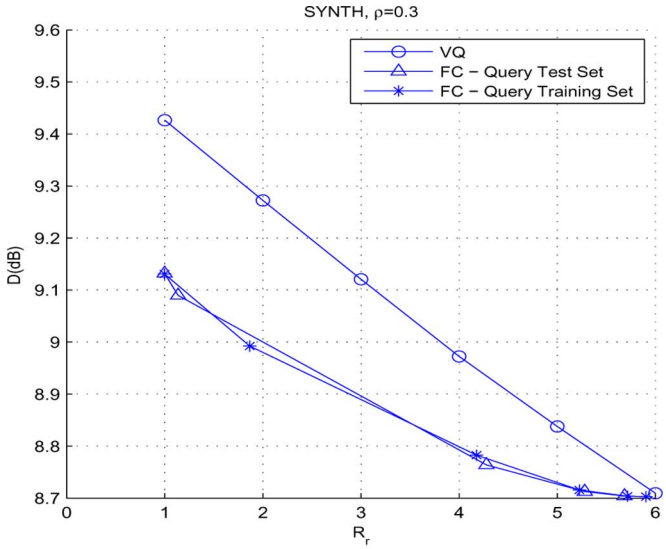

Fig. 9. Data-set SYNTH, $\rho = 0.3$ with query quantization.

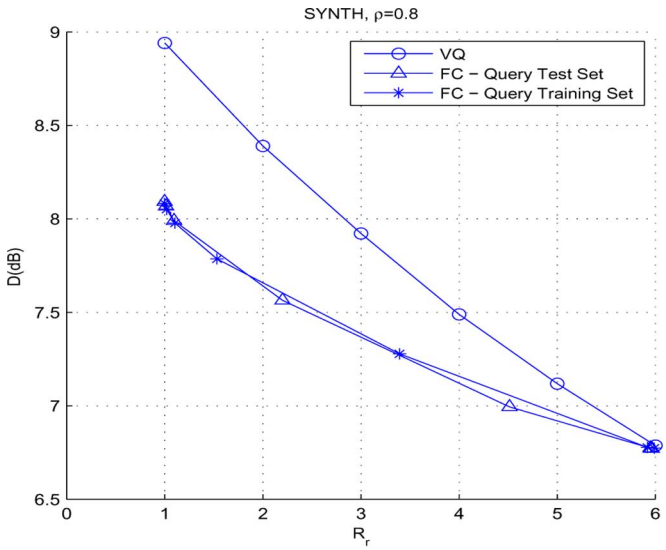

Fig. 10. Data-set SYNTH, $\rho = 0.8$ with query quantization.

at an average retrieval rate of 3 bits. In the Intel Berkeley Lab data-set (Fig. 8), we notice gains in the range of 2.6 dB at an average retrieval rate of 2 bits and 3.5 dB at an average retrieval rate of 3 bits. FC also provides $\approx 2\times$ reduction in retrieval rate at a distortion of 26.2 dB. We also note that increasing R_s results in better performance of the selective retrieval technique. This is possible since increasing storage allows more freedom in the design of the bit-selector. However, this increase in gain is relatively marginal in the real data-set. We believe that this is because the design algorithm gets trapped in local minima that riddle the cost surface.

Also note the different scaling in distortion for the real data-sets due to the presence of sources with high variance. For example, in the sensor data-set, voltage is measured in volts and varies in the range 2–3 V. However, light falling on the sensor is measured in lux and varies from 1 lx in moonlight, 400 lx in a bright office, and 100 000 lx in full sunlight.[3] The mean signal energy for the retrieval of nine sources is 57.65 dB, which equals the distortion for the $R_r = 0$ bits case. Thus, compression with FC and VQ leads to almost 5 dB reduction in distortion when $R_r = 6$ bits. For the STOCKS dataset, the mean signal energy for the retrieval of nine sources is 34.8 dB, and compression with FC and VQ leads to almost 14 dB reduction in distortion when $R_r = 6$ bits.

Finally, note that the "acceptable"/"good" distortion levels are dependent on particular applications and uses. As the allowed storage rate is increased, the distortion would decrease for both FC and VQ. However, (as is evident from experimental results) the performance would be similar, i.e., at $R_r = R_s$, the performance of FC and VQ are the same. Yet, as R_r is gradually reduced, VQ is forced to reduce R_s as well and, hence, has higher distortion than FC for the same R_r.

[3]See http://db.csail.mit.edu/labdata/labdata.html for details.

D. Quantization of the Query-Space

Once the queries were grouped as described in Section VII-A, we partitioned the query space with a decision tree [15], [16]. We preferred the decision tree over the nearest neighbor/nearest prototype classifiers since the performance of the latter would be dependent on meaningful distance/distortion metrics for the discrete query space $\{0,1\}^M$. For example, if two sources are highly correlated, the request for either or both of them should have the same retrieval cost (subset of bits), and hence it would make sense that these queries be handled together. It might be necessary to employ a (linear/nonlinear) transformation to a secondary feature space for the Euclidean norm to make sense. On the other hand, decision trees can operate on discrete data and are known to be efficient classifiers in this setting. Hence, we expect the decision tree paradigm to handle (unknown) queries.

The training set of queries were partitioned into $L = 6$ groups during the fusion coder design (as explained in Section VII). Subsequently, a decision tree that performs this classification was constructed, and the fusion coder components were reoptimized. Next, a test set of 345 queries was extracted from the same distribution. This new set of queries was classified by the decision tree, and the resulting distortion-retrieval rate performance of the system was evaluated.

We note a very small loss in performance of the system on the new set of queries as compared (see Figs. 5, 6, 9, and 10), and the performance advantages over the joint compression (VQ) scheme are maintained.

IX. Conclusion

We introduced the problem of fusion storage of correlated sources with selective retrieval. We proposed a *fusion coding* framework to perform optimal encoding, retrieval, and decoding of correlated sources. We presented the necessary conditions for optimality and proposed an iterative algorithm for fusion coder design, which is guaranteed to converge to a locally optimal solution. We observed that the proposed FC provides significant improvement in retrieval speed, at a prescribed distortion level as well as significantly better data reproduction quality, for a given retrieval speed, over both real and synthetic data-sets. Heuristics for "clever" bit-selector initialization and operational rate-distortion curve computation were also discussed.

The FC design complexity and the storage requirements of the bit-selector grow linearly with query-set size. By exploiting properties of the optimal FC and by performing quantization/partitioning of query space, this complexity growth can be eliminated. However, FC design complexity grows exponentially with the storage rate R_s compounding scalability to large storage rates and networks. Future work would focus on methods that trade design complexity against performance so that FC designs can scale to large sensor networks.

References

[1] D. Slepian and J. Wolf, "Noiseless coding of correlated information sources," *IEEE Trans. Inf. Theory*, vol. IT-19, no. 4, pp. 471–480, Jul. 1973.

[2] A. D. Wyner and J. Ziv, "The rate-distortion function for source coding with side-information at the decoder," *IEEE Trans. Inf. Theory*, vol. IT-22, no. 1, pp. 1–11, Jan. 1976.

[3] S. Pradhan and K. Ramchandran, "Distributed source coding using syndromes (DISCUS): Design and construction," in *Proc. DCC*, 1999, pp. 158–167.

[4] X. Liu and H. Ferhatosmanoglu, "Efficient k-NN search on streaming data series," in *Proc. SSTD*, 2003, pp. 83–101.

[5] J. Nayak, S. Ramaswamy, and K. Rose, "Correlated source coding for fusion storage and selective retrieval," in *Proc. ISIT*, 2005, pp. 92–96.

[6] T. Han and K. Kobayashi, "A unified achievable rate region for a general class of multiterminal source coding systems," *IEEE Trans. Inf. Theory*, vol. IT-26, no. 3, pp. 277–288, May 1980.

[7] S. Ramaswamy, J. Nayak, and K. Rose, "Code design for fast selective retrieval of fusion stored sensor network/time series data," in *Proc. ICASSP*, 2007, vol. 2, pp. 1005–1008.

[8] E. Tuncel, P. Koulgi, and K. Rose, "Rate-distortion approach to databases: Storage and content-based retrieval," *IEEE Trans. Inf. Theory*, vol. 50, no. 6, pp. 953–967, Jun. 2004.

[9] R. Weber, H. Schek, and S. Blott, "A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces," in *Proc. VLDB*, Aug. 1998, pp. 194–205.

[10] D. G. Luenberger, *Linear and Non-Linear Programming*. Reading, MA: Addison-Wesley, 1984.

[11] T. M. Cover and J. A. Thomas, *Elements of Information Theory*. New York: Wiley, 1991.

[12] A. Gersho and R. M. Gray, *Vector Quantization and Signal Compression*. Norwell, MA: Kluwer, 1992.

[13] K. Rose, "Deterministic annealing for clustering, compression, classification, regression, and related optimization problems," *Proc. IEEE*, vol. 86, no. 11, pp. 2210–2239, Nov. 1998.

[14] R. Cristescu and M. Vetterli, "On the optimal density for real-time data gathering of spatio-temporal processes in sensor networks," in *Proc. IPSN*, 2005, pp. 159–164.

[15] R. Duda, P. Hart, and D. G. Stork, *Pattern Classification and Scene Analysis*. New York: Wiley, 1973.

[16] J. Quinlan, "Induction of decision trees," *Mach. Learn.*, vol. 1, no. 1, pp. 81–106, 1986.

Sharadh Ramaswamy (M'08) received the B.Tech. and M.Tech. degrees from the Indian Institute of Technology, Madras, India, in 2003 and the Ph.D. degree from the University of California, Santa Barbara, in 2009.

He then joined MayaChitra, Inc., Santa Barbara, CA, as a Member of Research Staff. His research interests lie in compression, sensor networks, search and retrieval in databases, video processing, and machine learning.

Jayanth Nayak (M'08) received the Ph.D. degree in electrical and computer engineering from the University of California, Santa Barbara, in 2005.

He is currently a Research Staff Member with Mayachitra, Inc., Santa Barbara, CA. His research interests include information theory, pattern recognition, and computer vision.

Kenneth Rose (S'85–M'91–SM'01–F'03) received the Ph.D. degree from the California Institute of Technology, Pasadena, in 1991.

He then joined the Department of Electrical and Computer Engineering, University of California at Santa Barbara, where he is currently a Professor. His main research activities are in the areas of information theory and signal processing and include rate-distortion theory, source and source-channel coding, audio and video coding and networking, pattern recognition, search and retrieval in databases, and nonconvex optimization. He is interested in the relations between information theory, estimation theory, and statistical physics, and their potential impact on fundamental and practical problems in diverse disciplines.

Prof. Rose was co-recipient of the 1990 William R. Bennett Prize Paper Award of the IEEE Communications Society, as well as the 2004 and 2007 IEEE Signal Processing Society Best Paper awards.