On the Hardness of Finding Optimal Multiple Preset Dictionaries

Michael Mitzenmacher, *Member, IEEE*

***Abstract*—We show that the following simple compression problem is NP-hard: given a collection of documents, find the pair of Huffman dictionaries that minimizes the total compressed size of the collection, where the best dictionary from the pair is used to compress each document. We also show the NP-hardness of finding optimal multiple preset dictionaries for LZ'77-based compression schemes. Our reductions make use of the catalog segmentation problem, a natural partitioning problem. Our results justify heuristic attacks used in practice.**

***Index Terms*—Huffman coding, LZ'77, NP-completeness, preset dictionaries, two-stage compression.**

I. Introduction

Preset dictionaries are often used to improve compression. For example, with standard two-pass Huffman coding, one generally sends a table describing the encoding, or a *dictionary*, that allows the decoder to determine the appropriate codewords for each alphabet symbol. Instead, if similar transmissions occur on a repeated basis, a preset dictionary can be set in advance to avoid the cost of computing and transmitting an explicit dictionary each time. Avoiding memory and computation costs for dictionary computation may be useful even if it yields slightly worse compression. Preset dictionaries may also yield improved compression results when the cost of sending an explicit dictionary would be more than the gain the explicit dictionary would yield over the preset dictionary. This situation may occur when documents are short and a suitably effective preset dictionary can be found. Preset dictionaries arise in for example fax transmission and JPEG encoding [1].

A natural extension to this idea is to allow multiple preset dictionaries. Flag bits at the beginning of a file can be used to denote which (if any) preset dictionary to use. Allowing multiple dictionaries can improve compression, at the cost of more space to store the preset dictionaries and more computation to test which dictionary should be used for compression. Note that this additional computation is required only at the compression end, and is easily parallelized. Also, when the data consists of files of different types, choosing the best dictionary may be a simple task. The ZLIB library, designed for LZ'77-based compression [2], also allows for multiple preset dictionaries [3].

An early application of this idea dates back to the Voyager spacecraft, which coded blocks of 16 pixels using the best of four fixed preset memoryless entropy codes [4], [5]. This technique is generally called two-stage coding, where the first stage requires choosing an appropriate dictionary and the second stage uses the chosen dictionary to encode the data. For more recent variations of two-stage coding and an excellent treatment of the problem, see [6], [7].

Once multiple preset dictionaries are allowed, a natural question is whether optimal multiple preset dictionaries can be found efficiently. That is, we have a natural complexity question, similar in spirit to works such as [8], [9]. In this correspondence, we relate the problem of finding optimal multiple preset dictionaries to the model of *segmentation problems* introduced in [10]. In the spirit of these results, we refer to problems related to finding multiple preset dictionaries as compression segmentation problems. Using this connection, we show that natural compression segmentation problems for Huffman trees and LZ'77-based compression are NP-hard. Since previous work on two-stage compression algorithms have generally used heuristic techniques in order to determine dictionaries, our result can be seen as a strong justification for these heuristic attacks.

Notation and Terminology

We make note of the following notation, used throughout this correspondence. The notation $|\Sigma|$ is used for the size of a set Σ. A nonnegative function $f(x)$ is said to be $O(g(x))$ if and only if there exist positive constants c and N so that $f(x) \leq cg(x)$ for all $x \geq N$ on the domain of f. Similarly, a nonnegative function $f(x)$ is said to be $\Omega(g(x))$ if and only if there exist positive constants c and N so that $f(x) \geq cg(x)$ for all $x \geq N$ on the domain of f. All logarithms have base 2.

II. The Catalog Segmentation Problem

The problem of finding optimal families of preset dictionaries is related to the segmentation problems defined by Kleinberg, Papadimitriou, and Raghavan. The canonical segmentation problem is the *catalog segmentation* problem, which we first describe informally. A seller can send a catalog to all customers in its database. Only r items can be advertised in a catalog. Given previous history, the seller can exactly tell which people will buy which items. The goal is to maximize the number of sales. If the seller could create just one catalog, the optimal solution would be to include the r most popular items. Suppose instead the seller can create k different catalogs and send exactly one to each customer. How should the seller determine the k catalogs that will maximize the number of sales?[1]

Following [10], we formally define the catalog segmentation problem as follows. Consider the customers as sets of items $S_1, S_2, \ldots, S_n$ over a ground set U. Catalogs $X_1, X_2, \ldots, X_k$ are also sets of items. The goal is to choose the X_i such that $|X_i| \leq r$ for all i and

$$\sum_{j=1}^{n} \max_{1 \leq i \leq k} (|X_i \cap S_j|)$$

is maximized.

Theorem 1: **[10]** The catalog segmentation problem is NP-hard (even for $k = 2$).

Even though the catalog segmentation problem is NP-hard, it can be solved in polynomial time for any fixed r and k, since there are only $\binom{|U|}{r}$ possible catalogs. The brute-force algorithm of trying all combinations of k catalogs of r items is exponential in r and k.

Although in [10] the authors say that the catalog segmentation problem (and several natural variants) are NP-hard, complete proofs are not given. For completeness, in the Appendix , we offer our own simple proof of Theorem 1 for the case $k = 2$, suggested to us by

Manuscript received March 13, 2003; revised March 31, 2004. This work was supported in part by an Alfred P. Sloan Research Fellowship and by the National Science Foundation under Grants CCR-9983832, CCR-0118701, and CCR-0121154. The material in this correspondence was presented in part at the Data Compression Conference, Snowbird, UT, March 2001.

The author is with the Division of Engineering and Applied Sciences, Harvard University, Cambridge, MA 02138 USA (e-mail: michaelm@eecs.harvard.edu).

Communicated by W. Szpankowski, J. P. Kieffer, and E.-h. Yang, Guest Editors.

Digital Object Identifier 10.1109/TIT.2004.830778

[1]One can form a more general version of the catalog segmentation problem where the number of catalogs can vary, and there is a cost associated with the number of catalogs produced [10]. In this case, determining the proper number of catalogs is also part of the problem. For our hardness results we do not need this more general version of the problem.

Steve Lumetta. We reduce the catalog segmentation problem to the problems of finding optimal multiple preset dictionaries for Huffman coding and Lempel–Ziv coding, thereby showing that these problems are NP-hard. For convenience, for the remainder of the correspondence we focus on the case where $k = 2$, although our results are easily generalized to other values of k.

III. Huffman Coding

We now define the Huffman code segmentation problem. We are given a collection of documents $D_1, D_2, \ldots, D_n$ over an alphabet Σ. Finding an optimal sequence of Huffman code word lengths over Σ to compress these documents is trivial; it simply requires summing the character frequencies over all of the documents and using the standard Huffman tree algorithm. Suppose, however, we were allowed to construct k different Huffman dictionaries and use the best one to compress each document. The Huffman code segmentation problem is to minimize the total compressed size given the D_i and $k \geq 2$.

To see how the Huffman code segmentation problem might naturally arise, suppose we plan to design multiple preset Huffman dictionaries for a large, arbitrary collection of documents, such as all Web pages. We might then sample n representative pages as a test set in order to develop our Huffman dictionaries, which would be used over the larger class of documents. The Huffman code segmentation problem designs the best set of k dictionaries for this test set. Note that this problem is completely offline; that is, all the relevant pages are assumed to be available when choosing the dictionaries.

Like the catalog segmentation problem, the Huffman code segmentation problem has at its heart a clustering question. Instead of deciding what items need to go in each catalog, we need to decide which documents are associated with each preset dictionary. Formalizing this similarity yields our reduction.

Theorem 2: The Huffman code segmentation problem is NP-hard.

Proof: We reduce from catalog segmentation for the case $k = 2$. Recall for the catalog segmentation problem we have a ground set U with $|U| = m$ and n subsets $S_1, \ldots, S_n$ of U. We wish to find two subsets X and Y of U with size r such that

$$\sum_{j=1}^{n} \max(|X \cap S_j|, |Y \cap S_j|)$$

is maximized. We design a related Huffman code segmentation problem so that each element in the ground set corresponds to a character of Σ, and each character has depth d or $d+1$ for some d in the pair of optimal Huffman trees. The sets X and Y will correspond to the characters of depth d derived from elements of U in each Huffman tree.

More specifically, let d be the smallest integer such that $2^{d+1} \geq m + r$. Our alphabet Σ will consist of $2^{d+1} - r$ characters. The first m characters, $u_1, u_2, \ldots, u_m$, represent characters that correspond to elements of U. We also introduce additional characters $v_1, v_2, \ldots, v_h$, where $h = 2^{d+1} - r - m$, so that there are $2^{d+1} - r$ total characters. This setup allows each character to have depth d or $d+1$ in each of the optimal trees.

For each set S_j we construct a corresponding document D_j. The document D_j will contain three occurrences of each character u_q such that item q is contained in S_j, and two occurrences of every other character.

With this construction, we may assume without loss of generality that all of the characters $v_1, v_2, \ldots, v_h$ should have depth at least as large as any character u_i in both of the Huffman trees in the solution, because their frequency is at most as large in every document. Similarly, if the depths of all characters in both trees are not within one of each other, the total compressed size can be improved by flattening the offending tree. That is, if some node has depth a and two other nodes have depth (at least) $a+2$, we may improve the tree by replacing it with one where all three nodes have depth $a+1$. This reduces the size by at least $4n - 3n > 0$.

Hence, there must be exactly r characters from U with depth d in each of the two trees of the solution, and all other characters have depth $d+1$. We show that the sets of r characters with depth d in the two trees yield the sets X and Y for the catalog segmentation problem, by replacing characters with the corresponding elements. The size of D_j compressed using the optimal pair of Huffman trees is the sum of the following terms: $2(d+1)h$ for characters $v_1, v_2, \ldots, v_h$; $3d(\max(|X \cap S_j|, |Y \cap S_j|))$ for characters u_i in S_j of depth d in the better tree; and $3(d+1)(m - \max(|X \cap S_j|, |Y \cap S_j|))$ for other characters u_i. Hence, the total compressed size over the n documents is

$$2n(d+1)h + 3n(d+1)m - 3\sum_{j=1}^{n} \max(|X \cap S_j|, |Y \cap S_j|).$$

Minimizing the compression is, therefore, equivalent to maximizing the result of the catalog segmentation problem. □

The corresponding decision version, which asks if there is a pair of trees that compresses the documents down to t total bits, is clearly NP-complete.

Because optimal answers are NP-hard, it is natural to consider approximation algorithms. One observation is that using one Huffman tree is at most $\lceil \log k \rceil$ bits per character worse than using k Huffman trees. This follows because given the optimal Huffman trees for a given k, we could design a compression scheme where the first $\lceil \log k \rceil$ bits would specify which of the k trees to use, and the remaining bits would correspond to the appropriate codeword from that tree; the optimal single Huffman tree performs better than this solution. We suspect this bound can be improved. In practice, however, heuristic techniques for the catalog segmentation problem as discussed in [10] or other natural heuristic techniques can be applied and prove quite effective.

IV. Preset Dictionaries for an LZ'77 Scheme

The ZLIB format was primarily designed for use with the DEFLATE procedure, an LZ'77-based algorithm [3]. Since the LZ'77 format is standard and described fully in most basic compression texts (e.g., [1]), we rely on an informal description here. As a document is sequentially compressed (or decompressed), there is a window into the previous stream of characters. The current sequence of characters can be compressed by providing a pointer into the window of the previous character stream and a length denoting how many characters starting from that pointer are the same as the current stream. The decompressor can use these pointers to efficiently reconstruct the original text. In this setting, a preset dictionary consists of a sequence of characters that the compressor and decompressor use as an implicit prefix to the stream to be compressed. No output is generated while the compressor runs on this prefix; similarly, the decompressor is initialized by using the preset dictionary as an implicit prefix. As an example, mail messages contain common header fields such as "Subject," "From," and "Return-Path." Including these strings in a preset dictionary used to compress mail messages may therefore improve compression.

There are many variations of LZ'77, covering issues such as where to match in the window if multiple matches of different lengths are possible and how to encode the pointers. The first issue does not arise for our result. For simplicity, we adopt a model where any pointer and any match length use $\lceil \log s \rceil$ bits if s the size of the dictionary string. Also, generally a 1-bit prefix is used for each symbol in the encoding to mark whether it represents a pointer–length pair or a character from

the original document. Again, for simplicity, we ignore this in the proof below; it does not change the analysis. Our result extends naturally to other models.

Unlike the Huffman coding setting, for the LZ'77 setting even a single optimal preset dictionary of a fixed size s for a given set of documents can be NP-hard under appropriate models. For example, the related problem of determining if a collection of words can be condensed into a dictionary string S of size s in such a way that every word appears as a substring of S is NP-complete; it is the shortest common superstring problem [11]. Notice that this result relies on overlapping strings, and there may be unusual subtleties under other cost models related to how pointers are encoded and whether there are multiple matches. In practice, a natural approach for English text is to find the most frequently used words and use them as the basis for a dictionary, ignoring overlap issues or simply throwing out words that significantly overlap others.

Our focus here is quite different. We consider the LZ'77 segmentation problem: given $k \geq 2$ and a set of documents $D_1, D_2, \ldots, D_n$ over an alphabet Σ to determine the k best preset dictionaries of size at most s, where the size of D_i compressed is taken to be the minimum number of bits over the choice of the k dictionaries. We show that when $k \geq 2$, the problem is NP-hard. In our proof, all words use completely distinct characters, so this complexity does not arise from issues related to words with overlap. In this case, the heart of the matter is again a clustering question.

Theorem 3: The LZ'77 segmentation problem is NP-hard.

Proof: We again reduce from the catalog segmentation problem for $k = 2$. Given a catalog segmentation problem, we construct an LZ'77 segmentation problem whose alphabet Σ has size $z|U|$ for a value of z to be determined. (For notational convenience, we assume $|\Sigma|$ is a power of two henceforth.) For each u_i in the ground set $|U|$ we associate z distinct characters from Σ so that the characters associated with each u_i are disjoint; more specifically, we associate u_i with a length z string of the form $w_1 w_2 \ldots w_z$. For each set $S_j, j = 1, \ldots, n$, of the catalog segmentation problem, there is an associated document D_j constructed by concatenating all the words associated with the elements of S_j. (Notice that the order of the words in S_j will prove unimportant because of our choice of cost function for representing pointers and matches.) We seek dictionaries with size rz.

Let us temporarily assume that the optimal preset dictionaries consist of concatenated strings of length z, with each such string corresponding to the string corresponding to some u_i. Then the dictionaries correspond to subsets X and Y of the ground set U of size r in the natural way, and X and Y will be the optimal catalogs for the segmentation problem. When compressed, the size of document D_j will be

$$|S_j| z \log |\Sigma| - \max(|X \cup S_j|, |Y \cup S_j|)(z \log |\Sigma| - 2\lceil \log rz \rceil).$$

To see this, note that representing all the characters of D_j without compression takes $|S_j| z \log |\Sigma|$ bits. When compressing, we save $z \log |\Sigma|$ bits for each matched word between the document and the dictionary, although we incur a cost of $2\lceil \log rz \rceil$ for the relevant pointers describing the location and length of the match. Hence the total compressed size can be rewritten as

$$\sum_{j=1}^{n} \left[z \log |\Sigma| (|S_j| - \max(|X \cup S_j|, |Y \cup S_j|)) + 2\lceil \log(rz) \rceil \max(|X \cup S_j|, |Y \cup S_j|) \right].$$

With these conditions, the compression gain for each document is proportional (up to lower order terms) to the number of strings in the document that are matched in the dictionary. When z is sufficiently large compared to $\lceil \log(rz) \rceil$, the optimal solution to the LZ'77 segmentation problem maximizes

$$\sum_{j=1}^{n} \max(|X \cup S_j|, |Y \cup S_j|).$$

Clearly, z can be chosen to be polynomial in the size of the problem. The solution to the LZ'77 problem naturally yields a corresponding optimal solution to the catalog segmentation problem. Each dictionary maps to a catalog by mapping length z strings corresponding to some u_i in the dictionaries to items in the catalogs.

We return to the assumption that dictionaries consist of concatenated strings of length z, with each such string corresponding to the string corresponding to some u_i. We show that if we obtain optimal dictionaries that are not of this form, we can transform them to optimal dictionaries of this form (in polynomial time). Consider an optimal pair of dictionaries not of this form. For each dictionary, placing characters from a string corresponding to a u_i adjacent to each other in the appropriate order can only lower the compressed size, so we assume that this is done. Next, if a dictionary contains characters from a string corresponding to a u_i, but not the full z characters, find the substring with this property that is used most frequently in compressing the documents. Simply replace characters from a less frequently used (or equally used) substring with this property in the same dictionary (note that there must be one). This can only reduce the compressed size. Repeating this as necessary we can transform any optimal solution into another optimal solution with the desired property, in polynomial time. □

The key points in the above proof are that the compression changes the word cost from $\Omega(z)$ to $O(\log z)$, and that strings in the optimal dictionaries are concatenated words of length z. The proof works for any representation of pointers and match lengths that preserves these properties.

APPENDIX

We now give the proof of Theorem 1 in the case where $k = 2$.

Theorem 1: The catalog segmentation problem is NP-hard for $k = 2$.

Proof: We reduce from the well-known NP-hard problem Graph Bisection [12]: given a graph $G = (V, E)$ with an even number of vertices, split V into two disjoint sets V_1 and V_2 with $|V_1| = |V_2| = |V|/2$ such that the number of edges adjacent to both V_1 and V_2 is minimized. We turn an instance of graph bisection into a catalog segmentation problem as follows. For each vertex, create a corresponding item. If d is the maximum degree of the graph, create for each item $d + 1$ customers who want to purchase only that item. For each edge, create a customer that wants to purchase only those two items corresponding to the vertices adjacent to that edge. Now suppose we can have $r = |V|/2$ items in each catalog. We argue that the optimal pair of catalogs must contain all $|V|$ items. Otherwise, some item appears in both catalogs, but since the maximum degree of the graph is d, replacing one copy of the repeated item by some item that does not appear improves the number of items sold, as there are at least $d + 1$ customers that will purchase only that item. Now consider a pair of catalogs containing all $|V|$ items. Each such pair of catalogs corresponds to a bisection of the vertices into two disjoint sets V_1 and V_2 of size $|V|/2$. If c is the number of edges that are adjacent to both a vertex in V_1 and a vertex in V_2, then the the number of sales corresponding to the pair of catalogs is $(d+1)|V| + 2|E| - c$. We may conclude that the pair of catalogs that maximizes sales also provides a bisection that minimizes the number of edges crossing from V_1 to V_2. This completes the reduction. □

ACKNOWLEDGMENT

The author thanks the reviewers and the Guest Editors for their many suggestions.

REFERENCES

[1] I. Witten, A. Moffat, and T. Bell, *Managing Gigabytes*, 2nd ed. San Francisco, CA: Morgan Kaufmann, 1999.
[2] J. Ziv and A. Lempel, "A universal algorithm for data compression," *IEEE Trans. Inform. Theory*, vol. IT-23, pp. 337–343, May 1977.
[3] P. Deutsch and J-L. Gailly, "ZLIB Compressed Data Format Specification Version 3.3," Network Working Group, RFC 1950, 1996.
[4] R. F. Rice and J. R. Plaunt, "The Rice Machine: Television Data Compression," Jet Propulsion Laboratory, Pasadena, CA, Tech. Rep. 900-408, 1970.
[5] ——, "Adaptive variable-length coding for efficient compression of spacecraft television data," *IEEE Trans. Commun. Technol.*, vol. COM-19, pp. 889–897, Dec. 1971.
[6] P. Chou, M. Effros, and R. Gray, "A vector quantization approach to universal noiseless coding and quantization," *IEEE Trans. Inform. Theory*, vol. 42, pp. 1109–1138, July 1996.
[7] M. Effros, P. Chou, and R. Gray, "Weighted universal image compression," *IEEE Trans. Image Processing*, pp. 1317–1329, Oct. 1999.
[8] M. Garey, D. Johnson, and H. S. Witsenhausen, "The complexity of the generalized Lloyd-Max problem," *IEEE Trans. Inform. Theory*, vol. IT-28, pp. 255–256, Mar. 1982.
[9] J. A. Storer, "Data compression: Methods and complexity issues," Ph. D. dissertation, Princeton Univ., Princeton, NJ, 1979.
[10] J. Kleinberg, C. Papadimitriou, and P. Raghavan, "Segmentation problems: A micro-economic view of data mining," in *Proc. 30th ACM Symp. Theory of Computing*, 1998, pp. 473–482.
[11] M. Garey and D. Johnson, *Computers and Intractibility*. San Francisco, CA: Freeman, 1979.
[12] M. Garey, D. Johnson, and L. Stockmeyer, "Some simplified NP-complete graph problems," *Theor. Comput. Sci.*, vol. 1, no. 3, pp. 237–267, 1976.

Monotonicity-Based Fast Algorithms for MAP Estimation of Markov Sequences Over Noisy Channels

Xiaolin Wu, *Senior Member, IEEE*, Sorina Dumitrescu, and Zhe Wang

***Abstract*—In this correspondence, we study algorithmic approach to solving the problem of maximum *a posteriori* (MAP) estimation of Markov sequences transmitted over noisy channels, which is also known as the MAP decoding problem. For the class of memoryless binary channels that produce independent substitution and erasure errors, the MAP sequence estimation problem can be formulated and solved as one of the longest path in a weighted directed acyclic graph. But for algorithm efficiency, we transform the graph problem to one of matrix search. If the underlying matrix is totally monotone, then the complexity of MAP sequence estimation can be greatly reduced. We give a sufficient condition for the matrix induced by MAP sequence estimation to be totally monotone, which is indeed the case if the input sequence is Gaussian Markov. Under this condition, the complexity of MAP decoding can be reduced from $O(N^2M)$ to $O(NM)$, where N is the size of source alphabet and M is the length of input sequence. Furthermore, for Markov sequences of fixed-length code we propose a block parsing strategy to reduce the complexity of MAP sequence estimation to $O(M + N^2M/\log M)$ or to $O(M + NM/\log M)$, depending on if the total monotonicity holds.**

Another significance of this correspondence lies in the applicability of the presented algorithmic approach, which has been thoroughly studied in computer science literature, to many other discrete optimization problems encountered in both source and channel coding, ranging from optimal multiresolution and multiple-description quantizer design, to context quantization for minimum conditional entropy, and to optimal packetization with uneven error protection.

***Index Terms*—Gaussian Markov source, joint source–channel decoding, maximum *a posteriori* (MAP) sequence estimation, Monge inequality, string parsing, weighted directed acyclic graph.**

I. INTRODUCTION

We consider the problem of maximum *a posteriori* (MAP) estimation of an input source sequence with memory, possibly entropy coded, and transmitted over a noisy channel, or commonly known as the MAP decoding problem. The goal is to utilize the memory in the input sequence to correct or/and alleviate transmission errors without using error-correction codes. A MAP decoder tries to maximize the *a posteriori* probability of the decoded sequence, based on the statistics of both source and channel [6], [11]–[16]. It is a joint source–channel decoding technique that exploits the residual redundancy of the source-code stream.

The most common structure of source redundancy to be exploited by a MAP decoder is that of a Markov sequence. If a Markov sequence is coded by a fixed-length code then the MAP decoding is quite straightforward. The problem gets more complex if the source code is of variable length. This is because channel errors can easily cause loss of synchronization on a variable-length code (VLC). Since most entropy codes are of variable length, MAP decoding of variable-length-coded Markov sequence is of greater practical interest and importance and is the key problem to be solved in this work.

Typically, the MAP decoder employs a look-ahead mechanism, and makes a delayed or soft decision to take advantage of the memory in the input sequence. The common look-ahead mechanism in communication engineering is trellis. Indeed, all existing MAP decoding algorithms for VLCs are essentially modified Viterbi decoders [6], [11]–[16]. Most of these algorithms were designed for binary-symmetric channels (BSC).

In this correspondence, the MAP decoding problem for Markov sources is represented by a weighted directed acyclic graph (WDAG), which is more general and flexible than a trellis. We consider the memoryless binary channels that produce independent substitution and erasure errors, or so-called errors-and-erasure channels (EEC). The MAP decoding problem for EEC becomes one of computing the single-source longest path in the WDAG, and hence, can be solved exactly by the standard longest path algorithm. However, since our graph is quite dense and since the time complexity of the longest path algorithm is proportional to the number of edges, more efficient solutions are needed to make MAP decoding practical. This need motivated our research of algorithmic nature.

The purpose of this correspondence is two-fold. First, we present a new algorithmic approach to MAP decoding of VLC-coded Markov sequences. Our key idea is to transform the problem to one of matrix search. Via the transform, we expose and exploit a strong and useful monotonicity of the optimization objective function for MAP decoding, and drastically reduce the complexity of the problem. Second, while

Manuscript received May 11, 2003; revised March 22, 2004. This work was supported by Natural Sciences and Engineering Council of Canada and by the National Science Foundation. The material in this correspondence was presented in part at the IEEE Information Theory Workshop, Paris, France, April 2003.

The authors are with the Department of Electrical and Computer Engineering, McMaster University, Hamilton, ON L8S 4K1, Canada (e-mail: xwu@mail.ece.mcmaster.ca; sorina@mail.ece.mcmaster.ca; zwang@grads.ece.mcmaster.ca).

Communicated by E.-h. Yang, Guest Editor.
Digital Object Identifier 10.1109/TIT.2004.830782