Endcoding Complexity Versus Minimum Distance

Louay M. J. Bazzi and Sanjoy K. Mitter, *Life Fellow, IEEE*

***Abstract*—A bound on the minimum distance of a binary error-correcting code is established given constraints on the computational time–space complexity of its encoder where the encoder is modeled as a branching program. The bound obtained asserts that if the encoder uses linear time and sublinear memory in the most general sense, then the minimum distance of the code cannot grow linearly with the block length when the rate is nonvanishing, that is, the minimum relative distance of the code tends to zero in such a setting. The setting is general enough to include nonserially concatenated turbo-like codes and various generalizations. Our argument is based on branching program techniques introduced by Ajtai. The case of constant-depth AND-OR circuit encoders with unbounded fanins are also considered.**

***Index Terms*—Binary codes, branching programs, encoding complexity, minimum distance, time–space tradeoffs.**

I. Introduction

A BINARY error-correcting code is specified by an injective map $\mathcal{E} : \{0,1\}^n \rightarrow \{0,1\}^m$ from the binary strings of length n to binary strings of length m. The map $\mathcal{E}$ is called the *encoding map*, the image $\mathcal{C}$ of $\mathcal{E}$ in $\{0,1\}^m$ is called the *code*, n is called the *message length*, m is called the *block length*, the elements of $\mathcal{C}$ are called the *codewords*, and the ratio n/m is called the *rate* of the code. A fundamental parameter that characterizes the worst case error-correction capabilities of the code is its *minimum distance* which is defined as the minimum Hamming distance between two distinct codewords. One of the main goals in combinatorial coding theory is to find codes with a good trade off between rate and minimum distance, as the message length tends to infinity. A code (meaning a family of codes indexed by the block length) is called *an asymptotically good code* if the block length grows linearly with the message length and the minimum distance grows linearly with the block length. Otherwise, the code is called *asymptotically bad*.

In this paper, we consider the following question:

> What can be said about the growth of the minimum distance of a binary code given constraints on the computational complexity of its encoder?

We concentrate mainly on the time–space complexity of the encoder. In this setting, the above question is a natural tradeoff between the parameters: minimum distance, rate, encoding time, and encoding space.

From a practical perspective, this question is important since there are popular error-correcting codes that have low time–space encoding complexity. We are referring here to turbo codes, or more precisely, to parallel concatenated turbo codes introduced by Berrou, Glavieux, and Thitimajshima in [4], and repeat–convolute codes introduced by Divsalar, Jin, and McEliece in [6], [7]. This low time–space encoding complexity is crucial for the corresponding iterative decoding algorithms. These decoding algorithms use the state-space representation of the encoder and their running time is proportional to the cardinality of the state space. Sharp bounds on the minimum distance of turbo codes were first obtained by Kahale and Urbanke [9], [10] for random interleavers and constant memory convolutional codes. In a recent joint paper with Mahdian and Spielman [3], we derived strong bounds on the minimum distance of turbo-like codes in a variety of cases. One of these cases is the well-structured setting of generalized repeat–convolute codes where the convolutional code is replaced by an arbitrary automaton. We argued that such codes are asymptotically bad when the memory of the automaton is sublinear and the number of repetitions is constant.

In this paper, this particular result is extended to the much more general setting where the encoder is a binary branching program, or equivalently, a nonuniform random-access machine with binary input registers.

A general theorem is established that asserts that if the encoder is a binary branching program that uses linear time and sublinear space, then the minimum distance of the code cannot grow linearly with the block length when the rate is nonvanishing, which is a rather surprising result. In general, a bound is derived relating the involved parameters.

Our proof is based on the branching program techniques introduced in the recent paper of Ajtai [1].

Also considered is the case of constant-depth AND-OR circuit encoders with unbounded fanins. We conclude with a conjecture about the strongest possible time–space tradeoffs that can be obtained for encoding asymptotically good codes.

II. Branching Program Encoders

A *branching program* $\mathcal{B}$ (binary, by default, i.e., two-way) is defined by a connected, directed acyclic graph with a single source and a single sink, together with a set of n binary *input variables* and a set of m binary *output variables* which satisfy the following. There are exactly two edges leaving each non-sink node, the first labeled with a one input-label and the second with a zero input-label. Every non-sink node is associated with an input variable. Some of the edges of the graph are associated with output variables (possibly more than one output

Manuscript received May 7, 2003; revised March 3, 2005. This work was supported by the Army Research Office under MURI Grant: Data Fusion in Large Arrays of Microsensors DAAD19-00-1-0466, the Department of Defense MURI Grant: Complex Adaptive Networks for Cooperative Control Subaward 03–132, and the National Science Foundation under Grant CCR-0325774.

L. M. J. Bazzi is with the Department of Electrical and Computer Engineering, the American University of Beirut (AUB), Beirut, Lebanon (e-mail: Louay.Bazzi@aub.edu.lb).

S. K. Mitter is with the Department of Electrical Engineering and Computer Science, The Massachusetts Institute of Technology (MIT), Cambridge, MA 02139 USA (e-mail: mitter@mit.edu).

Communicated by R. Urbanke, Associate Editor for Coding Techniques.

Digital Object Identifier 10.1109/TIT.2005.847727

variable per edge), in which case the edges are labeled by binary output-labels (a separate 0 or 1 for each output variable associated with the edge). The nodes of the graph are called *states*, its source is called the *start state*, and its sink is called the *end state*. The branching program $\mathcal{B}$ computes an encoding map $\mathcal{E}: \{0,1\}^n \to \{0,1\}^m$ as follows. The computation starts by reading the value of the variable associated with the start state and moving, according to its value, to the next state, and so on, by reading more bits until the end state is reached. At each transition along an edge, the output variables associated with the edge (if any) are set according to the edge output labels. We may want to assume that on any input, each output variable will be set at least once, or we can assume that the output variables are arbitrarily preset. The *computation of the branching program on an input* is the corresponding sequence of states starting with the start state and ending with the end state.

More formally, by a *branching program* $\mathcal{B}$ we mean a 5-tuple $\mathcal{B} = (n, m, G, in\text{-}var, out\text{-}val)$, where we have the following.

- n and m are positive integers.
- $G = (V, E)$ is a connected, directed acyclic graph. V is the set of vertices of the graph and E is its set of edges. The graph G has a single source and a single sink. The elements of V are called *states*. The source node of G is called the start state and is denoted by $\text{start} \in V$. The sink node of G is called the end state and is denoted by $sink \in V$. The graph G has exactly two edges $e_{s,0}$ and $e_{s,1}$ leaving each nonsink node $s \in V\backslash\{sink\}$. Thus, E is in one-to-one correspondence with $(V\backslash\{sink\})\times\{0,1\}$.[1] G is associated with the state transition map

$$trans : (V\backslash\{sink\}) \times \{0,1\} \to V$$

  given by $e_{s,b} = (s, trans(s,b))$ for each $s \in V$ and for each $b \in \{0,1\}$.
- $in\text{-}var$ is the the input-variables map

$$in\text{-}var : V\backslash\{sink\} \to [n].$$

  (If i is an integer, we mean by $[i]$ the set $\{1,\ldots,i\}$.)
- $out\text{-}val$ is the output-values map

$$out\text{-}val : E \to \cup_{A\subset[n]}\{0,1\}^A.$$

  The output-values map is associated with the output-variables map $out\text{-}var : E \to 2^{[m]}$ which is specified by $out\text{-}val(e) \in \{0,1\}^{out\text{-}var(e)}$ for all $e \in E$.

The branching program $\mathcal{B}$ *computes* an encoding map $\mathcal{E}: \{0,1\}^n \to \{0,1\}^m$ as follows.

The *computation of the branching program* $\mathcal{B}$ *on an input* $x \in \{0,1\}^n$ is the sequence of states $s_0, s_1, \ldots, s_{\ell-1}$, where $s_0 = start, s_{\ell-1} = sink$, and $s_i = trans(s_{i-1}, x_{in\text{-}var(s_{i-1})})$ for $i = 1,\ldots,\ell-2$. Note that the end state must be reached after a finite number of steps since the graph G is acyclic.

The *output* $y = \mathcal{E}(x)$ *of the branching program* $\mathcal{B}$ *on the input* x is a string in $\{0,1\}^m$ defined by $y = y^{(\ell-1)}$, where $y^{(0)}, \ldots, y^{(\ell-1)}$ are defined iteratively as: $y^{(0)}$ is the all-zeros strings in $\{0,1\}^m$, and $y^{(i)}$ is derived from $y^{(i-1)}$ via

$$y_j^{(i)} = \begin{cases} y_j^{(i-1)}, & \text{if } j \notin out\text{-}var(s_{i-1}, s_i) \\ out\text{-}val(s_{i-1}, s_i)_j, & \text{if } j \in out\text{-}var(s_{i-1}, s_i) \end{cases}$$

for $j = 1,\ldots,m$ and $i = 1,\ldots,\ell-1$.

The *length of the computation of* $\mathcal{B}$ *on* x is defined as its number of states, i.e., ℓ. The *time of the computation of* $\mathcal{B}$ *on* x is defined as its number of states plus the total number of times each output variable is set during the computation of $\mathcal{B}$ on x, i.e.,

$$\ell + \sum_{i=1}^{\ell-1} |out\text{-}var(s_{i-1}, s_i)|.$$

Also, the branching program $\mathcal{B}$ *encodes* the code $\mathcal{C}$, which is defined as the image of $\mathcal{E}$ in $\{0,1\}^m$. $\mathcal{B}$ is also called a *branching program encoder* for the code $\mathcal{C}$.

Finally, we associate three complexity measures with a branching program: length, time, size, and memory (or space). The *length* of the branching program $\mathcal{B}$ is the maximum length of a computation of $\mathcal{B}$ on an input x where the maximization is over all input strings $x \in \{0,1\}^n$. Similarly, the *time* of the branching program is the maximum time of a computation. The *size* S of the branching program is its number of states, i.e., $S = |V|$. The *memory* or the *space* M of a branching program is defined as $M = \log_2 S$.

A. Random-Access Machines as Encoders

The codes encodable by such general branching programs correspond to those encodable by *nonuniform random-access machines* with binary input registers which we define below.

The branching program model was introduced by Borodin and Cook [5] as a model of random-access machines for studying time–space tradeoffs for sorting algorithms; see also Ajtai [1].

In this subsection, the relation between the two models is explained in the context of encoders. A *nonuniform random-access machine encoder* is an 8-tuple

$$\mathcal{M} = (n, m, V, start, sink, trans, in\text{-}var, out\text{-}val)$$

where

- n and m are positive integers;
- V is a finite set of states;
- $start$ and $sink$ are special states in V called the start and end states, respectively;
- $trans$ is the state transition map $trans : (V\backslash\{sink\}) \times \{0,1\} \to V$.
  A graph $G = (V, E)$ is associated with $trans$ whose vertex-set is the set of states V, and whose set E of edges is constructed as follows. For each nonsink state $s \in V\backslash\{sink\}$, there are two edges $(s, trans(s,0))$ and $(s, trans(s,1))$ in E labeled with 0 and 1, respectively.
  It is assumed that the state transition map satisfies the condition that G is an acyclic graph with $start$ as its single source vertex and $sink$ as its single sink vertex.

[1] Since the edges are labeled, multiple edges are allowed, i.e., the possibility that $e_{s,0}$ and $e_{s,1}$ have the same destination is allowed.

- *in-var* is the the input-variables map $in\text{-}var : V\backslash\{sink\} \to [n]$.
- *out-val* is the output-values map $out\text{-}val : (V\backslash\{sink\}) \times \{0,1\} \to \cup_{A\subset[n]}\{0,1\}^A$.
  The output-values map is asociated with the output-variables map $out\text{-}var : (V\backslash\{sink\}) \times \{0,1\} \to 2^{[m]}$ which is specified by $out\text{-}val(s,b) \in \{0,1\}^{out\text{-}var(s,b)}$ for all $s \in V$ and $b \in \{0,1\}$.

The equivalence of the random-access machine model and the branching program model of an encoder is obvious from the definition. The machine $\mathcal{M}$ functions in exactly the same way as the underlying branching program $\mathcal{B}$, and its *time* and *memory* are defined as those of $\mathcal{B}$.

Note that this model, being nonuniform, does not restrict the size of the code description. This contributes to the generality of the results in this paper since we are deriving lower bounds. One consequence of this unrestricted code description size is that any code can be encoded in linear time and linear space. This can be done using an exponential-size code description as explained in Example 2.4.

In the rest of the paper, we will be working with the branching program model.

B. Some Special Types of Branching Programs

The branching program is called *leveled* if the states are divided into an ordered collection of sets each called a *level* where edges are between consecutive levels only. In such a case, the *width* of the branching program is the maximum number of states per level.

The branching program is called *oblivious* if the input variables (and the output variables) are read (respectively, set) in the same order regardless of the input under consideration. Thus, an oblivious branching program is naturally leveled in such a way that all the nodes in the same level read the same input variables, and set the same output variables when moving to the next level.

The branching program is called a *read-k-times* branching program if each input variable is read at most k times on any input.

The branching program is called a *write-w-times* if at most w output variables are set per transition.

C. Examples

Example 2.1: The trellis of a rate-$1/w$ convolutional code is an oblivious, read-once, write-w-times branching program. Since we decided for simplicity to allow branching program encoders to have only one end state, we are assuming here that the states in the last level of the trellis are grouped into a single state. The width of the branching program is 2^{M_0}, where M_0 is the number of memory registers of the convolutional code. The size of the branching program is $n2^{M_0}+1$, where n is the block length. The length of the branching program is $n+1$ and its time is $(w+1)n+1$. The additional 1 is needed to account for the end state.

Note that not any oblivious read-once branching program encoder is a trellis of some convolutional code since such a branching program can encode a nonlinear code. Moreover, the graph structure of such a branching program need not be uniform in the sense that the transitions structure can change from one level to the other.

Example 2.2: Parallel-concatenated turbo codes are encodable by low-complexity oblivious branching programs as follows.

A *parallel-concatenated turbo encoder* [4] $\mathcal{E}$ with a constant number k of branches, message length n, and memory M_0 is specified by k permutations $\pi_1,\dots,\pi_k$, each on n bits and a rate-1 convolutional code Q (the *component code*) with M_0 memory registers. For x in $\{0,1\}^n, \mathcal{E}(x)$ is encoded as $\mathcal{E}(x) = (x, Q(\pi_1(x)),\dots,Q(\pi_k(x)))$, where $\pi_i(x)$ is the string obtained by permuting the bits of x according to the permutation π_i, and $Q(y)$ is the output of the convolutional encoder Q on the input string y. $\mathcal{E}$ is naturally encodable by an oblivious read-k-times write-2-times branching program $\mathcal{B}$ as follows. Let $\mathcal{B}_1$ be the read-once write-2-times branching program corresponding to the trellis of the rate-$1/2$ systematic convolutional code $\mathcal{E}_1(x) = (x, Q(\pi_1(x))$ (see Example 2.1). For $i = 2,\dots,k$, let $\mathcal{B}_i$ be the read-once write-once branching program corresponding to the trellis of the convolutional code $\mathcal{E}_i(x) = Q(\pi_1(x))$. $\mathcal{B}$ is the concatenating of $\mathcal{B}_1,\dots,\mathcal{B}_k$ resulting from identifying the end state of $\mathcal{B}_i$ with the start state of $\mathcal{B}_{i+1}$ for $i = 1,\dots,k-1$. The start state of $\mathcal{B}$ is an element of $\mathcal{B}_1$ and its end state is an element of $\mathcal{B}_k$.

Thus, $\mathcal{B}$ has length $kn+1$ and time $(2k+1)n+1$. The width of $\mathcal{B}$ is 2^{M_0}, and its size is at most $kn2^{M_0}+1$.

Example 2.3: Repeat–convolute codes fit in the same picture. A *repeat–convolute code* [6], [7] consists of a repeat-k -times code, a convolutional code, and a permutation. More precisely, a repeat–convolute encoder $\mathcal{E}$ of message length n and memory M_0 is specified by a constant integer k, a permutation π on kn bits, and a rate-1 convolutional encoder Q with M_0 memory registers. For x in $\{0,1\}^n$, $\mathcal{E}(x)$ is encoded as $\mathcal{E}(x) = (x, Q(\pi(r(x))))$, where r is the repeat-k-times map, i.e., $r(x)$ is the concatenation of k copies of x.

$\mathcal{E}$ is naturally encodable by a leveled, oblivious, read-k-times, write-2-times, length-$(kn+1)$, time-$((2k+1)n+1)$, and width-2^{M_0} branching program $\mathcal{B}$ whose size is at most $kn2^{M_0}+1$.

The branching program $\mathcal{B}$ is constructed as follows. Let $\mathcal{B}'$ be the read-once write-2-times branching program corresponding to the trellis of the rate-$\frac{k}{k+1}$ convolutional encoder $\mathcal{E}'(y) = (y_1,\dots,y_n,Q(y))$, where $y \in \{0,1\}^{kn}$. $\mathcal{B}$ is constructed from $\mathcal{B}'$ by relabeling the input variables of $\mathcal{B}'$ according to the repetition map r and the permutation π.

Example 2.4: Any binary code can be trivially encoded in the binary branching program model in linear time and linear space by a leveled and oblivious branching program $\mathcal{B}$ whose graph has a tree structure. The graph of $\mathcal{B}$ consists of a height-n complete binary tree whose leaves are grouped into a single sink node. The root of the tree is the start state of $\mathcal{B}$. The binary input labels of the edges correspond to the left and right labels of the binary tree edges. Thus, the simple paths from the root of the tree to the sink node are in one-to-one correspondence with the input strings in $\{0,1\}^n$. On any such input string, the branching program follows the corresponding path and outputs the whole

codeword at the last transition from the nth level to the sink node.

This linear encoding time and space makes sense for all codes in the random-access machine encoding model because we are not restricting the amount of read-only memory, which is exponential in this example, needed to store the code description (see Section II-A).

Example 2.5: Any block-length-n binary linear code is naturally encodable by a leveled, oblivious, width-2, $O(n^2)$-time, write-once branching program $\mathcal{B}$. The reason is that one can multiply (over GF(2)) a binary input vector by a fixed binary matrix using such a branching program. The length of $\mathcal{B}$ is $D+1$, where D is the number of ones in the matrix. The time of $\mathcal{B}$ is $D+1+n=O(n^2)$.

III. Main Result

Theorem 3.1: Let $\mathcal{E}:\{0,1\}^n \to \{0,1\}^m$ be an injective encoding map computable by a branching program $\mathcal{B}$ of size $S=S(n)$, time $t=t(n)$, and length $\ell=\ell(n)$.

Let $\mathcal{C}$ be the binary code associated with $\mathcal{E}$, i.e., the image of $\mathcal{E}$ in $\{0,1\}^m$.
If $t(n)=\Theta(n)$, then the minimum distance of $\mathcal{C}$ is

$$O\left(\left(\frac{\log_2 S}{n}\right)^{\frac{1}{\lfloor \ell/n \rfloor}} n\right).$$

Therefore, $\mathcal{C}$ is asymptotically bad when $S(n)=2^{o(n)}$ and $t(n)=O(n)$. More generally, if $t(n)=\Omega(n)$, then the minimum distance of $\mathcal{C}$ is

$$O\left(\left(\frac{t}{n}\right)^3 \left(\frac{\log_2 S}{n}\right)^{\frac{n}{2t}} n\right).$$

Thus, $\mathcal{C}$ is also asymptotically bad when $S(n)=2^{O(n^{1-\epsilon_1})}$ and

$$t(n)=O(n\log_2^{1-\epsilon_2} n), \qquad \text{for all } \epsilon_1,\epsilon_2>0.$$

In other words, linear time and sublinear space for encoding imply that the code is asymptotically bad, i.e., the minimum distance cannot grow linearly with the block length when the rate is nonvanishing.

A. Application to Turbo-Like Codes

By applying the first bound in Theorem 3.1 to parallel concatenated turbo codes and repeat–convolute codes (see Examples 2.2 and 2.3), we can recover the corresponding bound in [3] as follows.

The minimum distance of a parallel concatenated turbo code with a constant number k branches, message length n, and M_0 memory registers is $O(n^{1-1/k}(M_0+\log_2 n)^{1/k})$ because the length ℓ of the corresponding branching program is $\ell=kn+1$ and its size S is at most $kn2^{M_0}+1$ (see Example 2.2), and hence $\lfloor \ell/n \rfloor = k$ and $\log_2 S = O(M_0+\log_2 n)$. Note that this bound is slightly weaker than the corresponding bound $O(n^{1-1/k}M_0^{1/k})$ in [3]. To get rid of the additive $\log_2 n$ term, apply Theorem 4.1 below, which is stated in terms of the width $W=2^{M_0}$ of the branching program. Similarly, the minimum distance of a repeat–convolute code with k repetitions, message length n, and M_0 memory registers is $O(n^{\ell-1/k}M_0^{1/k})$.

So both types of codes will be asymptotically bad as long as M_0 is sublinear in n. Note that the situation when M_0 is sublinear in n corresponds to the case when the underlying trellis has subexponential size, i.e., when the corresponding iterative turbo decoding algorithm has subexponential running time.

IV. Proof of Theorem 3.1

A. Ajtai Proof Techniques for the Hamming Distance Problem

To prove the theorem, we use branching program techniques introduced by Ajtai in [1]. More specifically, we are referring to these branching program techniques to show that there is no $O(n)$-time and $o(n\log_2 n)$-space R-way branching program, where $R=n^c$ and c is some absolute constant that solves the Hamming distance problem: given n strings in $\{0,1\}^{\log_2 R}$, decide whether any two of them that are distinct are at Hamming distance $\lambda \log_2 R$ apart, where λ is another absolute constant related to c.

Even though this is a decision problem in the setting of R-way branching programs, while ours is not a decision problem and is in the setting of two-way branching programs, the techniques introduced by Ajtai lie behind the proof described later.

The reader is referred to Ajtai's paper [1] for further details.

B. Objects and Terminologies Under Consideration

We start by making the branching program $\mathcal{B}$ leveled. Recall from Section II: this means that the states are partitioned into ℓ consecutive sets $L_0,\ldots,L_{\ell-1}$ of states, each called a level in such a way that edges, i.e., transitions, occur only between consecutive levels.

We will divide (meaning, partition) $\mathcal{B}$ into *blocks* (meaning, sets) of consecutive levels, i.e., sets of the form $B(i,j)=\{L_i,L_{i+1},\ldots,L_j\}$. For a given block $B(i,j)$, we will be looking at states in the *lower boundary level of the block*; meaning, the last level in the block with respect to the ordering of the levels in the block $B(i,j)$, i.e., L_j.

Given an input, we will be looking at the *computation* of the branching program $\mathcal{B}$ on this input, which, as explained in Section II, is defined to be the corresponding sequence of states starting with the start state and ending with the end state. So, in the leveled case, each computation takes exactly ℓ steps, i.e., it contains exactly ℓ states.

Fix an input string x, and consider the computation $\{s_i\}_i$ of the branching program $\mathcal{B}$ on x. Fix also a set L of levels or a set T of blocks.

By an *input bit or variable x_j being accessed or read* in L (or T) *during the computation* of $\mathcal{B}$ on x, we mean that there is a state s_{i_0} in the computation $\{s_i\}_i$ of $\mathcal{B}$ on x that belongs to a level in L (or to a level in a block in T) such that the value of the input variable x_j is read in order to move from s_{i_0} to s_{i_0+1}, i.e., $j=in\text{-}var(s_{i_0})$ in the terminology of Section II.

Similarly, by an *output bit or variable y_j being set* in L (or T) *during the computation* of $\mathcal{B}$ on x we mean that there is a state s_{i_0} in the computation $\{s_i\}_i$ of $\mathcal{B}$ on x that belongs to a level in L (or to a level in a block in T) such that the value of

the output variable y_j is set at the transition form s_{i_0} to s_{i_0+1}, i.e., $j \in out\text{-}var(s_{i_0}, s_{i_0+1})$ in the terminology of Section II.

Finally, by a computation containing a sequence of states, we mean that each state in this sequence appears in the computation. Note that here the order does not matter since the states in a computation are distinctly due to the fact that the branching program graph is acyclic.

C. The Oblivious Case Argument

Recall that an oblivious branching program is naturally leveled in such a way that all the nodes in the same level read the same input variables, and set the same output variables.

Since the Proof of Theorem 3.1 is relatively long, it is instructive to look first at the very special case when $\mathcal{B}$ is oblivious. This case is very restrictive compared to a general branching program. To restrict the setting further, assume that $\mathcal{B}$ is read-k-times and write-w-times, where $k = O(1)$ and $w = O(1)$. This additional restriction will further simplify the argument.

In [3], the setting of automata is used to bound the minimum distance of repeat–convolute codes. More specifically, we studied the case of a repeat–convolute code where the convolutional code is replaced by an arbitrary automaton. Even though the automata setting is less general than the case being considering here, the argument can naturally be extended as follows.

Theorem 4.1: Let $\mathcal{E} : \{0,1\}^n \to \{0,1\}^m$ be an injective encoding map computable by a read-k-times, write-w-times, width-W oblivious branching program $\mathcal{B}$, where $k = O(1)$ and $w = O(1)$.

Let $\mathcal{C}$ be the binary code associated with $\mathcal{E}$, i.e., the image of $\mathcal{E}$ in $\{0,1\}^m$.

Assume that $W \geq 2$. Then the minimum distance of $\mathcal{C}$ is

$$O\left(n\left(\frac{\log_2 W}{n}\right)^{1/k}\right).$$

Therefore, $\mathcal{C}$ is asymptotically bad when $W = 2^{o(n)}$.

Note that this bound is slightly sharper than that of Theorem 3.1 since it is in terms of the width W of the branching program which is smaller than its size S.

Note also that the time t of the branching program in Theorem 4.1 is at most wkn, i.e., $t = \Theta(n)$. Thus, the more general case when t is superlinear in n, which is handled in the second bound in Theorem 3.1, is not applicable in the setting of Theorem 4.1.

Proof: We will exhibit two distinct input strings that map to two codewords at distance $O(n(\log_2 W)/(n)^{1/k})$. We will do this by finding a nonempty set of input variables U, a subset J of levels, and two distinct strings x_1 and x_2 in $\{0,1\}^n$ such that x_1 and x_2 agree outside U, and the computations of $\mathcal{B}$ on x_1 and x_2 agree outside J. This will give us the desired bound on the minimum distance. J will be constructed as a union of intervals from a partition of $\mathcal{B}$ that we define next.

Let ℓ be the length of the branching program, and note that $n \leq \ell \leq kn$, where the first inequality follows from the fact that each input variable must be read in at least one level, since $\mathcal{E}$ is injective.

Partition $\mathcal{B}$ into b consecutive blocks, each consisting of p_1 or p_2 levels, where $p_1 = \lfloor \ell/b \rfloor$ and $p_2 = \lceil \ell/b \rceil \leq \lceil kn/b \rceil$. Assume for now that b is arbitrary as long as $1 \leq b \leq \ell$. We will optimize on the integer b later.

Each of the n input variables is read by $\mathcal{B}$ in at most k blocks. Recall that $\mathcal{B}$ is oblivious. Thus, for any specific variable, these blocks will be the same irrespective of the setting of the input variables. Define a k-set of blocks to be a set of at most k blocks. There are at most b^k possible k-set of blocks. So there are at least n/b^k input variables that are read by $\mathcal{B}$ in the same k-set of blocks. Let U be such a set of input variables with $|U| = \lceil n/b^k \rceil$, T be such a k-set of blocks, thus, $1 \leq |T| \leq k$. The set J mentioned earlier is the union of the blocks in T.

Consider the lower boundary levels (see the definition in Section IV-B) $L_1, \ldots, L_{|T|}$ of the blocks in T ordered by the level index, and let Q be the set of strings in $\{0,1\}^n$ which are zero outside U, thus, $|Q| = 2^{|U|}$. There are at most W^k state sequences in $L_1 \times \cdots \times L_{|T|}$, and for each x in Q the computation of $\mathcal{B}$ on x contains such a sequence. So if we can guarantee that $2^{|U|} > W^k$, we understand that there should be a sequence of states $\{\mathcal{S}_i\}_{i=1}^{|T|}$ in $L_1 \times \cdots \times L_{|T|}$ and two different strings x_1 and x_2 in Q such that the computation of $\mathcal{B}$ on both x_1 and x_2 contains $\{\mathcal{S}_i\}_{i=1}^{|T|}$. Since x_1 and x_2 agree outside U, the computation of $\mathcal{B}$ on x_1 and x_2 are exactly the same outside the blocks in T. Thus, $\mathcal{E}(x_1)$ and $\mathcal{E}(x_2)$ can only differ in the blocks in T. This means that the distance between $\mathcal{E}(x_1)$ and $\mathcal{E}(x_2)$ is at most

$$|T| p_2 w \leq k\lceil kn/b \rceil w$$

since $|T| \leq k$ and $p_2 \leq \lceil kn/b \rceil$.

This bound holds under the assumption that $2^{|U|} > W^k$, which can be guaranteed if

$$2^{n/b^k} > W^k.$$

We choose

$$b = \left\lceil \left(\frac{n}{k \log_2 W}\right)^{1/k} \right\rceil - 1$$

to guarantee this requirement.

Note that the only other constraints on b are $1 \leq b \leq \ell$. Since $k \geq 1$ and $W \geq 2$, the chosen value of b satisfies $b < n \leq \ell$. The chosen value of b satisfies $b \geq 1$ when W is not exponential in n. Note also that if W is exponential in n (i.e., if $\log_2 W = \Theta(n)$), the statement of Theorem 4.1 is trivial.

By replacing the chosen value of b in the upper bound $k\lceil kn/b \rceil w$ on the distance between $\mathcal{E}(x_1)$ and $\mathcal{E}(x_2)$, we get

$$k\left\lceil \frac{kn}{\left\lceil \left(\frac{n}{k \log_2 W}\right)^{1/k} \right\rceil - 1} \right\rceil w = O\left(n\left(\frac{\log_2 W}{n}\right)^{1/k}\right)$$

since $k = O(1)$ and $w = O(1)$.

This is an upper bound on the minimum distance of $\mathcal{C}$ since $\mathcal{E}(x_1) \neq \mathcal{E}(x_2)$ because $x_1 \neq x_2$ and $\mathcal{E}$ is injective. □

This proof is short and simple. But when $\mathcal{B}$ is not oblivious, this proof does not go through. The main reason is that U and

T cannot be constructed regardless of the values the input variables assume, as shown above. Moreover, in the nonoblivious case, the read-k-times and the write-w-times restrictions become restrictive. For example, in the general branching program model, depending on the input, a very large number of the output variables may be fixed in a particular transition, or a particular input variable may be read a very large number of times.

The next subsection is a sketch of how to handle the general situation. The proof is longer and more sophisticated. This is to be expected since the statement we are proving is much more general. The reader is encouraged to go carefully over the previous argument before proceeding to the general case.

D. Proof Technique

This subsection is an informal overview of the main techniques used in the proof. The formal argument is in the following sections. These techniques were introduced by Ajtai [1] to study the Hamming distance problem.

We want to find two input strings x_1 and x_2 such that $\mathcal{E}(x_1)$ and $\mathcal{E}(x_2)$ are close to each other.

The first step is to make the branching program leveled without affecting its input–output behavior. Next, divide the branching program into blocks each consisting of consecutive levels whose number will be suitably selected later and whose sizes are as uniform as possible. To exhibit x_1 and x_2, find a set T of blocks such that:

- the size of T is small;
- the computations of $\mathcal{B}$ on x_1 and x_2 are exactly the same in the blocks outside T; and
- not too many output bits of $\mathcal{E}(x_1)$ (respectively, $\mathcal{E}(x_2)$) are set in any of the blocks in T during the computation of $\mathcal{B}$ on x_1 (respectively, x_2).

Thus, $\mathcal{E}(x_1)$ and $\mathcal{E}(x_2)$ can only disagree on the few output bits that are set in T.

To find such x_1, x_2, and T, first find T together with a set Q' of input strings in $\{0,1\}^n$ and a sequence $\{\mathcal{S}_i\}_i$ of states in the lower boundary levels of the blocks in T in such a way that for each x in Q'

- the computation of $\mathcal{B}$ on x contains $\{\mathcal{S}_i\}_i$;
- not too many output bits of $\mathcal{E}(x)$ are set in any of the blocks in T during the computation of $\mathcal{B}$ on x; and
- the number of variables in x that are accessed only in the blocks in T during the computation of $\mathcal{B}$ on x is large.

The desired x_1 and x_2 are eventually found inside Q' as follows.

Modify the branching program $\mathcal{B}$ again so that $\mathcal{B}$ is forced to to pass through a state in the sequence $\{\mathcal{S}_i\}_i$ each time it attempts to leave a lower boundary level of a block in T, but without affecting its input–output behavior on Q'.

Using T, define an equivalence relation on $\{0,1\}^n$ by relating two strings if

- they share the same set of input variables that are not read during the computation of $\mathcal{B}$ in blocks outside T and
- they agree on the values of their bits outside this set.

Thus, each equivalence class $[x]$ is determined by a set $I_{[x]}$ of input variables and a setting of the variables outside $I_{[x]}$.

The computation of $\mathcal{B}$ is forced to contain the states $\{\mathcal{S}_i\}_i$ on all inputs so that $|[x]| = 2^{|I_{[x]}|}$, and hence the size of each equivalence class $[x]$ can be guaranteed to be large when $I_{[x]}$ is large.

For each input string x in Q', the number of input variables that are accessed only in the blocks in T during the computation of $\mathcal{B}$ on x is by construction large. Hence, the equivalence class $[x]$ of each x in Q' is large.

By considering the set Ω of sufficiently large equivalence classes, such that the equivalence classes of all the elements of Q' are guaranteed to be elements of Ω, our problem reduces to selecting the number of blocks so that $|Q'|$ is strictly larger than $|\Omega|$, and hence there are distinct x_1 and x_2 in Q' that have the same equivalence class. The fact that $[x_1] = [x_2]$ means that the computations of $\mathcal{B}$ on x_1 and x_2 are exactly the same outside the blocks in T, and hence $\mathcal{E}(x_1)$ and $\mathcal{E}(x_2)$ can only disagree on the output bits that are set inside the blocks in T.

By construction, the number of those output bits will be small. Moreover, $\mathcal{E}(x_1)$ and $\mathcal{E}(x_2)$ are distinct since $\mathcal{E}$ is injective. The distance between $\mathcal{E}(x_1)$ and $\mathcal{E}(x_2)$ will be the desired bound on the minimum distance of $\mathcal{C}$.

E. Proof Steps

Assume for the moment that $t = \Theta(n)$. We will deal with the more general case when we are done by defining the constants more carefully. So say that

$$t = an \quad \text{and} \quad \ell = cn \tag{1}$$

where $a, c \geq 1$ are constants ($a, c \geq 1$ because $\mathcal{E}$ is injective).

- **Step I:** The branching program is modified to be leveled. This can be done by a classical procedure. Construct a leveled directed graph of ℓ levels where each level consists of a copy of all the nodes of the original branching program together with the related output labels. Connect the nodes in each two consecutive levels according to the graph of the original branching program. This results in an end state in each level. Associate each end state not contained in the last level with an arbitrary input variable and connect it to the end state in the next level by two edges, the first with zero-input-label and the second with one-input-label. Finally, remove all the nodes (together with the ingoing and outgoing edges) which are not accessible from the start state in the first level or cannot reach the end state in the last level. The start state of the new branching program is the remaining state in the first level and its end state is the remaining state in the last level.

  The modified branching program computes the same function, i.e., $\mathcal{B}$ computes $\mathcal{E}$. The length of the resulting branching program $\mathcal{B}$ is ℓ, its time is t, its size is at most $S\ell$, and its width is at most S. The difference is that edges are now only between consecutive levels, and each computation takes exactly ℓ steps.
- **Step II:** Partition $\mathcal{B}$ into b consecutive blocks, each consisting of p_1 or p_2 levels, where

$$p_1 \stackrel{\text{def}}{=} \left\lfloor \frac{\ell}{b} \right\rfloor = \left\lfloor \frac{cn}{b} \right\rfloor \quad \text{and} \quad p_2 \stackrel{\text{def}}{=} \left\lceil \frac{\ell}{b} \right\rceil = \left\lceil \frac{cn}{b} \right\rceil. \tag{2}$$

Assume for now that in general $1 \leq b \leq \ell$ so that $1 \leq p_1, p_2 \leq \ell$. The integer b will be optimized on later.

- **Step III:**
  Lemma 4.2: There exist
  a) absolute constants $h, \alpha > 0$;
  b) $Q' \subset \{0,1\}^n$ such that
  $$|Q'| \geq \frac{2^n}{(Sb)^k} \tag{3}$$
  where
  $$k \stackrel{\text{def}}{=} \lfloor c \rfloor; \tag{4}$$
  c) a set of blocks T
  $$1 \leq |T| \leq k; \tag{5}$$
  d) and a sequence $\{\mathcal{S}_i\}_{i=1}^{|T|}$ of states in the lower boundary levels of the blocks in T, such that for each x in Q':
  1) the computation of $\mathcal{B}$ on x contains $\{\mathcal{S}_i\}_i$,
  2) at most
  $$w \stackrel{\text{def}}{=} \frac{hp_1 t}{l} \tag{6}$$
  output bits of $\mathcal{E}(x)$ are set in each block in T during the computation of $\mathcal{B}$ on x, and
  3) the number of variables in x that are accessed only in the blocks in T during the computation of $\mathcal{B}$ on x is at least $(\alpha n)/(b^k)$.

  Proof: See Section IV-F. □
- **Step IV:** Now modify the branching program $\mathcal{B}$ again so that $\mathcal{B}$ is forced to to pass through a state in $\{\mathcal{S}_i\}_i$ each time it attempts to leave a lower boundary level of a block in T, while guaranteeing that $\mathcal{B}$ behaves exactly like the old $\mathcal{B}$ on the inputs in Q', i.e., it computes $\mathcal{E}(x)$ for each x in Q'.
  This can be done simply by connecting (on both inputs) all the states in the level above that of $\mathcal{S}_i$ to $\mathcal{S}_i$, for each i. Note that $\mathcal{B}$ need not compute an injective function anymore, so it may not read all the input variables on some inputs. It may also leave some of the output variables unset, but this is not a problem since it is assumed that the output variables were arbitrarily preset.
- **Step V:** Finally, in Section IV-G we bound the minimum distance of $\mathcal{C}$ by exhibiting distinct x_1 and x_2 in Q' such that the distance between $\mathcal{E}(x_1)$ and $\mathcal{E}(x_2)$ is $O(n((\log_2 S)/(n))^{\frac{1}{k}})$.
- **Step VI:** In Section IV-H, we explain how to drop the assumption $t = \Theta(n)$.

F. Proof of Lemma 4.2

Consider any input x in $\{0,1\}^n$.

- Let $k \geq 1$ be an integer, and let $h > 0$. Choose k, and then h to continue with the proof.
- Let R_x be the set of those blocks such that each of the boxes renders fixed
  $$w \stackrel{\text{def}}{=} \frac{hp_1 t}{l}$$
  bits of $\mathcal{E}(x)$ during the computation of $\mathcal{B}$ on x.
- Let D_x be the set of input variables that are read in at most k states during the computation of $\mathcal{B}$ on x.
- And let D'_x be the set of input variables in D_x that are read only in blocks in R_x during the computation of $\mathcal{B}$ on x.

First recall from (1) that $a, c \geq 1$ are the constants satisfying
$$t = an \quad \text{and} \quad \ell = cn.$$
Recall also from (2) that
$$p_1 \stackrel{\text{def}}{=} \left\lfloor \frac{\ell}{b} \right\rfloor = \left\lfloor \frac{cn}{b} \right\rfloor \geq 1 \quad \text{and} \quad p_2 \stackrel{\text{def}}{=} \left\lceil \frac{\ell}{b} \right\rceil = \left\lceil \frac{cn}{b} \right\rceil.$$
We have the following bounds.

- From the definition of R_x, $(w+1)(b - |R_x|) \leq t$ since each of the $b - |R_x|$ blocks not in R_x fixes at least $w+1$ output variables during the computation of $\mathcal{B}$ on x. Thus, $w(b - |R_x|) \leq t$, i.e.,
  $$|R_x| \geq b - \frac{t}{w} = b - \frac{l}{hp_1} \geq b\left(1 - \frac{2}{h}\right) \tag{7}$$
  where the equality follows from the definition of w as $w = hp_1 t/\ell$, and the last inequality follows from the bound $p_1 = \lfloor \ell/b \rfloor \geq \ell/(2b)$.
- From the definition of D_x, each of the $n - |D_x|$ input variables outside D_x must be read in at least $k+1$ states in the computation of $\mathcal{B}$ on x. Moreover, each input variable in D_x must be read in at least one state in the computation of $\mathcal{B}$ on x since $\mathcal{E}$ is injective. Thus,
  $$|D_x| + (k+1)(n - |D_x|) \leq \ell$$
  because the number of states in the computation of $\mathcal{B}$ on x is at most ℓ. We can rewrite this inequality as
  $$|D_x| \geq n\left(1 - \frac{\ell/n - 1}{k}\right) = n\left(1 - \frac{c-1}{k}\right)$$
  since $\ell = cn$. Thus, if we set
  $$k \stackrel{\text{def}}{=} \lfloor c \rfloor$$
  we get
  $$|D_x| \geq n(1-\epsilon), \qquad \text{where } \epsilon \stackrel{\text{def}}{=} \frac{c-1}{k} < 1. \tag{8}$$
- The number of input variables read in blocks outside R_x is at most
  $$(b - |R_x|)p_2 \leq \frac{2bp_2}{h} \leq n\frac{4c}{h}$$
  where the first inequality follows from (7), and the second from the bound $p_2 = \lceil cn/b \rceil \leq 2cn/b$. Thus, by the definition of D'_x, we must have
  $$|D'_x| \geq |D_x| - n\frac{4c}{h} \geq n\left(1 - \epsilon - \frac{4c}{h}\right)$$
  where the second inequality follows from (8).
  Let h be sufficiently large such that
  $$\alpha \stackrel{\text{def}}{=} 1 - \epsilon - \frac{4c}{h} > 0. \tag{9}$$
  Note that this implies also that $1 - \frac{2}{h} > 0$ since $c \geq 1$, i.e,
  $$|R_x| > 0$$
  by (7).

In summary, some constants $h, \alpha > 0$ have been fixed, and $k \stackrel{\text{def}}{=} \lfloor c \rfloor$ has been specified such that
$$1 \leq |R_x| \leq b \quad \text{and} \quad |D'_x| \geq \alpha n. \tag{10}$$

Now, keep the definition of R_x in mind, ignore D_x, and recall that D'_x is a set of input variables such that

- each input variable in D'_x is read in at most k levels during the computation of $\mathcal{B}$ on x and
- each of those levels belongs to a block in R_x.

Recall also that up to now an input x in $\{0,1\}^n$ has been fixed.

Consider all the k-sets in R_x, i.e., the subsets of R_x of cardinality at most k. Each input variable in D'_x is read in such a k-set during the computation of $\mathcal{B}$ on x (recall the definition of an input variable being read in a set of blocks from Section IV-B), and there are at most $|R_x|^k$ such k-sets, so there are at least

$$\frac{|D'_x|}{|R_x|^k} \geq \frac{\alpha n}{b^k}$$

input variables in D'_x that are read in the same k-set of blocks in R_x during the computation of $\mathcal{B}$ on x, where we have used (10) to obtain the estimate. Let U_x be such a set of variables in D'_x, and let T_x be such a k-set of blocks in R_x, such that

$$1 \leq |T_x| \leq k \quad \text{and} \quad |U_x| \geq \frac{\alpha n}{b^k}.$$

Note that T_x is nonempty since $\mathcal{E}$ is injective.

Associate each x in $\{0,1\}^n$ with any such a U_x and T_x.

There are at most b^k such T_x, so there is a subset $Q \subset \{0,1\}^n$ and a k-set of blocks T such that

$$|Q| \geq \frac{2^n}{b^k}$$

and $T = T_x$ for each x in Q. Now consider the lower boundary levels $L_1, \ldots, L_{|T|}$ of the blocks in T ordered by the level index. There are at most S^k state sequences in $L_1 \times \cdots \times L_{|T|}$, and for each x in Q, the computation of $\mathcal{B}$ on x contains such a sequence, so there is a sequence $\{\mathcal{S}_i\}_{i=1}^{|T|}$ of states in $L_1 \times \cdots \times L_{|T|}$ and a subset $Q' \subset Q$ such that

$$|Q'| \geq \frac{|Q|}{S^k} \geq \frac{2^n}{(Sb)^k}$$

and the computation of $\mathcal{B}$ on x contains $\{\mathcal{S}_i\}_i$, for each x in Q'.

This completes the Proof of Lemma 4.2. □

G. Step V: Bounding the Minimum Distance

We are now ready to find the two distinct messages x_1 and x_2 that are mapped by $\mathcal{E}$ to codewords that are close to each other.

Using T, for each x in $\{0,1\}^n$, let I_x be the set of input variables that are not read during the computation of $\mathcal{B}$ on x in blocks outside T. Note that a double negation is needed ("not read" and "outside") since some of the input variables may not be read at all because the branching program in Step IV was modified.

So, from (3) in Lemma 4.2, for each x in Q'

$$|I_x| \geq \frac{\alpha n}{b^k}. \tag{11}$$

Using T, define the equivalence relation $\sim$ on $\{0,1\}^n$ by $x \sim y$ if

- $I_x = I_y$ and
- x agrees with y on the bits outside I_x.

In other words, $x|_{I_{[x]}} = y|_{I_{[y]}}$, where $[x]$ means the equivalence class of x.

Given any x in $\{0,1\}^n$, each $y \sim x$ can only disagree with x on I_x. Conversely, if y disagrees with x only inside I_x, it must be the case that $y \sim x$. To see why this is true, note that in Step IV all the computations of $\mathcal{B}$ were forced to leave the blocks in T in the same states: the $\{\mathcal{S}_i\}_i$ that was exhibited in (1) in Lemma 4.2. So the computations of $\mathcal{B}$ on x and y are exactly the same outside the blocks in T, and hence, any bit accessed on x outside T will be accessed on y outside T and none of the bits in I_x will be accessed on y outside T. It follows that

$$|[x]| = 2^{|I_x|}.$$

Thus, by (11), for each x in Q'

$$|[x]| \geq 2^{\alpha n/b^k}.$$

Let Ω be the set of equivalence classes such that the size of each equivalence class is at least $2^{\alpha n/b^k}$. So, $[x]$ is in Ω for each x in Q'. Besides, since the equivalence classes are disjoint, $|\Omega| 2^{\alpha n/b^k} \leq 2^n$, i.e.,

$$|\Omega| \leq \frac{2^n}{2^{\alpha n/b^k}}. \tag{12}$$

If we can guarantee that

$$|Q'| > |\Omega| \tag{13}$$

it follows that there will exist $x_1 \neq x_2$ in Q' such that $[x_1] = [x_2]$. The fact that $[x_1] = [x_2]$ means that the computations of $\mathcal{B}$ on x_1 and x_2 are exactly the same outside the blocks in T, and hence, $\mathcal{E}(x_1)$ and $\mathcal{E}(x_2)$ can only disagree on the output bits that are set inside the blocks in T. But, by (2) in Lemma 4.2, Q' was constructed in such a way that for on any x in Q', each block in T can set at most w bits of $\mathcal{E}(x)$ during the computation of $\mathcal{B}$ on x. Thus, $\mathcal{E}(x_1)$ and $\mathcal{E}(x_2)$ can disagree on at most

$$2|T|w \leq 2k\frac{hp_1 a}{c} \leq \frac{2khcna}{bc} = \frac{2khan}{b} \tag{14}$$

bits, where the first inequality follows from $|T| \leq k$ (by (5)) and $w = hp_1 t/\ell = hp_1 a/c$ (by (6) and (1)), and the second follows from $p_1 = \lfloor \ell/b \rfloor = \lfloor cn/b \rfloor \leq cn/b$ (by (2)).

Moreover, $\mathcal{E}(x_1)$ and $\mathcal{E}(x_2)$ must disagree on at least one bit since $x_1 \neq x_2$, and $\mathcal{E}$ is injective.

Using (3) and (12), condition (13) can be guaranteed to hold if

$$2^{\alpha n/b^k} > (Sb)^k$$

which is fulfilled when

$$\frac{1}{b} > \left(\frac{k \log_2 (Scn)}{\alpha n}\right)^{1/k}$$

since $b \leq \ell = cn$. If b is selected so that this holds, the minimum distance of $\mathcal{C}$ is at most $2khan/b$. The only restriction on b is $1 \leq b \leq \ell$, so use

$$b \stackrel{\text{def}}{=} \left\lceil \left(\frac{\alpha n}{k \log_2 (Scn)} \right)^{1/k} \right\rceil - 1. \tag{15}$$

This is always less than ℓ, and it cannot be less than 1 unless $S \geq 2^{\alpha n/k 2^k - \log_2 (nc)}$ in which case the statement of the theorem is trivial. Thus, via (14), the minimum distance of $\mathcal{C}$ is at most

$$\frac{2khan}{\left\lceil \left(\frac{\alpha n}{k \log_2 (Scn)} \right)^{1/k} \right\rceil - 1} = O\left(n \left(\frac{\log_2 S}{n} \right)^{\frac{1}{k}} \right).$$

H. Step VI: Dropping the Linear Time Assumption

Now we drop the assumption that $t = \Theta(n)$. Thus, a and c need not be constants. Since $\ell \leq t$, we use a as an upper bound on c. Assume that a grows with n and assume also that it is $O(\log_2 n)$ since otherwise the statement of Theorem 3.1 is trivial. We will not choose $k = \lfloor c \rfloor$. Going back to (8) and (9), we have

$$\alpha = 1 - \frac{c-1}{k} - \frac{4c}{h} > 1 - \frac{a-1}{k} - \frac{4a}{h},$$

a value that is needed to keep bounded away from zero by a positive constant. Set $k = \lceil 2(a-1) \rceil$ and $h = \lceil 16a \rceil$, thus, $\alpha > 1/4$.

Using the same choice of b as in (15), and the same bound on the minimum distance of $\mathcal{C}$ in (14), but with the new values of k and h and the bound $c \leq a$, we obtain that the minimum distance of $\mathcal{C}$ is at most

$$\frac{2\lceil 2(a-1) \rceil \lceil 16a \rceil an}{\left\lceil \left(\frac{n/4}{\lceil 2(a-1) \rceil \log_2 (San)} \right)^{1/\lceil 2(a-1) \rceil} \right\rceil - 1} = O\left(a^3 n \left(\frac{\log_2 S}{n} \right)^{\frac{1}{2a}} \right).$$

V. When the Encoder Is a Constant-Depth and-or Circuit

To outline the boundaries of the "encoding complexity versus minimum distance" question studied in this paper, the same problem is considered, but from the perspective of the circuit complexity of the encoder. Here note that not much can be said other than what is essentially expected. Since we know from [11] that there are asymptotically good codes that are encodable by linear-size and logarithmic-depth circuits, we are left with constant-depth circuits encoders with unbounded fanins.

Let $\mathcal{C} \subset \{0,1\}^m$ be a code. We say that $\mathcal{C}$ is encodable by an unbounded-fanin depth-d AND-OR circuit if C has an encoder $\mathcal{E} : \{0,1\}^n \to \{0,1\}^m$ (i.e., $\mathcal{E}$ is an injective map such that $\mathcal{C} = Image(\mathcal{E})$) such that each of the m output variables $\mathcal{E}_1(x), \ldots, \mathcal{E}_m(x)$ is computable by an unbounded-fanin depth-d AND-OR circuit on the n input variables $x_1, \ldots, x_n$. We require the gates in the circuit to be AND or OR gates with possibly negated inputs, but we allow the number of inputs per gate to be unbounded. The size of the circuit is defined as the total number of gates.

We argue by a direct application of the Hastad switching lemma that a polynomial size constant-depth circuit cannot encode an asymptotically good code. We will actually show that a size-S constant-depth circuit cannot encode an asymptotically good code as long as $\log_2 S = o(m^{1/(\ell+1)})$, where ℓ is the depth of the circuit. This is not surprising since, in the special case of linear codes, a small-depth circuit encoder corresponds to a code with a low-density generator matrix.

Lemma 5.1 The Hastad Switching Lemma [8]: Let $f : \{0,1\}^n \to \{0,1\}$ be computable by an unbounded-fanin depth-d AND-OR circuit of size M. Consider a random restriction ρ that independently keeps each input bit unset with a probability $p = 1/(20\ k)^d$, sets it to 1 with a probability $(1-p)/2$, and to 0 with a probability $(1-p)/2$. Then the probability, over the choice of ρ, that f, when restricted to the values set by ρ, cannot be computed by a decision tree of depth k is at most $M2^{-2k}$.

Note that a decision tree computing a binary function $b : \{0,1\}^n \to \{0,1\}$ on n variables is a binary tree where each node is associated with one of the input variables, and each leaf is associated with a 0 or 1 setting of the single output variable. This implies that if any setting of the input variables are fixed, there are at most k variables which when negated will affect the value of b, where k is the depth of the tree. In other words, when k is small, b has low sensitivity. Thus, if a code $\{0,1\}^n \to \{0,1\}^m$ (an injective map) is encodable by m decision trees, each of depth k, a direct counting argument shows that its minimum distance can be at most km/n. The Hastad switching lemma essentially reduces the circuit case to this situation.

Theorem 5.2: Let $\mathcal{E} : \{0,1\}^n \to \{0,1\}^m$ be an injective encoding map computable by an unbounded fanin AND-OR circuit of size S and depth ℓ.

Let $\mathcal{C}$ be the binary code associated with $\mathcal{E}$, i.e., the image of $\mathcal{E}$ in $\{0,1\}^m$. Assume that $m = \Theta(n)$, i.e., assume that the code-rate is nonzero.

Then the minimum distance of $\mathcal{C}$ is

$$O((20)^\ell \log_2^{\ell+1} (mS)).$$

Thus, $\mathcal{C}$ is asymptotically bad when $\ell = O(1)$ and $S = 2^{o(m^{1/(\ell+1)})}$.

Proof: Let $x_1, \ldots, x_n$ be the input variables, $A_1, \ldots, A_m$ the circuits that compute the output variables $y_1, \ldots, y_m$, and $a = m/n$. Thus, $a \geq 1$ is constant, the size of each A_i is at most S, and the depth of each A_i is at most ℓ.

Hit the x_i's with a random restriction ρ that keeps each x_i unset with a probability $p = 1/(20k)^\ell$, sets x_i to 1 with a probability $(1-p)/2$, and to 0 with a probability $(1-p)/2$.

Then, for each A_i, from the Hastad switching lemma, the probability that A_i does not collapse to a decision tree of depth k is at most $S2^{-2k}$. Thus, the probability that one of the A_i's

does not collapse, or the number of remaining (unset) variables is less than $np/2$ is at most

$$P \stackrel{\text{def}}{=} mS2^{-2k} + \frac{np(1-p)}{(np/2)^2} = mS2^{-2k} + \frac{4(1-p)}{np}$$

where the second term comes from the Chebychev inequality.

Fix $k = \log_2(Sm)$ so that

$$P \leq 1/(Sm) + 4(20\log_2(Sm))^{\ell/n} < 1$$

when n is large enough and S is subexponential in $n^{1/\ell}$. Note that when S is exponential in $n^{1/\ell}$, the statement of the theorem is trivial.

So, fix any restriction ρ with the property that

- the set I of input variables left unset by ρ has size at least $np/2$ and
- each of the A_i's collapses under ρ to a decision tree T_i of depth k, where $k = \log_2(Sm)$ and $p = 1/(20k)^\ell$.

Consider any setting of the variables in I, and let I_i be the set of variables in I read by T_i on this setting. Each I_i contains at most k variables, and the output of T_i can only be affected when we change some of the variables in I_i. So there should be a variable in I that appears in at most

$$\frac{\sum_{j=1}^{m} |I_j|}{|I|} \leq \frac{km}{np/2} = \frac{2ka}{p}$$

of the I_i's. By flipping this variable, at most $2ka/p$ output bits can be affected, and at least one output bit since $\mathcal{E}$ is injective. Hence, the minimum distance of $\mathcal{C}$ is at most

$$\frac{2ka}{p} = \frac{2\log_2(Sm)a}{1/(20\log_2(Sm))^\ell} = O\left((20)^\ell \log_2^{\ell+1}(Sm)\right). \quad \square$$

VI. Open Questions

Using branching program techniques introduced by Ajtai [1], we argued in Theorem 3.1 that there are no asymptotically good codes that are encodable in linear time and sublinear space in the most general sense. Conversely, we know that there are asymptotically good codes that are encodable in linear time and linear space (e.g., Spielman [11] explicitly constructed such codes. See also Example 2.4.) Thus, when the encoding time is linear, the linear memory requirement is asymptotically tight for encoding asymptotically good codes.

On the other extreme, any linear code can be encoded by a quadratic time branching program that uses minimal memory (see Example 2.5).

We conjecture the following in general.

Conjecture 6.1: Let $\mathcal{E} : \{0,1\}^n \to \{0,1\}^m$ be an injective encoding map computable by a branching program of memory M and time T, where $MT = o(n^2)$.

Let $\mathcal{C}$ be the binary code associated with $\mathcal{E}$, i.e., the image of $\mathcal{E}$ in $\{0,1\}^m$. Assume that $m = \Theta(n)$, i.e., assume that the code-rate is nonzero.

Then the minimum distance of $\mathcal{C}$ must be $o(n)$, i.e., $\mathcal{C}$ cannot be asymptotically good.

Proving the conjecture or finding the correct time–space tradeoffs for encoding asymptotically good codes when the encoding time is superlinear and subquadratic is very desirable.

Acknowledgment

The authors would like to thank Daniel Spielman and Rüdiger Urbanke for many stimulating conversations on this material. The authors wish to thank the anonymous referees for their detailed and very careful reading of the paper which has significantly improved the clarity and precision of the exposition.

References

[1] M. Ajtai, "Determinism versus nondeterminism for linear time RAMs," in *Proc 31st Annu. ACM Symp. Theory of Computing (STOC)*, Atlanta, GA, May 1999, pp. 632–641.

[2] L. Bazzi, "Minimum distance of error correcting codes versus encoding complexity, symmetry, and pseudorandomness," Ph.D. dissertation, MIT, Cambridge, MA, 2003.

[3] L. Bazzi, M. Mahdian, and D. Spielman, "The minimum distance of turbo-like codes," *IEEE Trans. Inf. Theory*, 2004, to be published.

[4] C. Berrou, A. Glavieux, and P. Thitimajshima, "Near Shannon limit error-correcting coding and decoding: Turbo-codes," in *Proc., IEEE Int. Conf. Communications*, Geneva, Switzerland, May 1993, pp. 1064–1070.

[5] A. Borodin and S. Cook, "A time-space tradeoff for sorting on a general sequential model of computation," in *SIAM J. Comput.*, vol. 11, 1982, pp. 287–297.

[6] D. Divsalar, H. Jin, and R. J. McEliece, "Coding theorems for turbo-like codes," in *Proc. 36th Annu. Allerton Conf. Communication, Control and Computing*, Monticello, IL, Sep. 1998, pp. 201–210.

[7] H. Jin and R. J. McEliece, "Coding theorems for turbo code ensembles," *IEEE Trans. Inf. Theory*, vol. 48, no. 6, pp. 1451–1461, Jun. 2002.

[8] J. Hastad, "Computational limitations for small depth circuits," Ph.D. dissertation, MIT, Cambridge, MA, 1986.

[9] N. Kahale and R. Urbanke, "On the minimum distance of parallel and serially concatenated codes," *IEEE Trans. Inf. Theory*, to be published.

[10] ——, "On the minimum distance of parallel and serially concatenated codes," in *Proc. IEEE Int. Symp. Information Theory*, Cambridge, MA, Aug. 1998, p. 31.

[11] D. A. Spielman, "Linear-time encodable and decodable error-correcting codes," *IEEE Trans. Inf. Theory*, vol. 42, no. 6, pp. 1723–1732, Nov. 1996.