

A Simple Taboo-Based Soft-Decision Decoding Algorithm for Expander Codes

Albert M. Chan, *Student Member, IEEE*, and Frank R. Kschischang, *Member, IEEE*

Abstract—We present a simple soft-decision decoding algorithm that modifies Sipser and Spielman’s hard-decision sequential “bit-flipping” algorithm for decoding expander codes. The algorithm incorporates symbol reliability information and a simple “taboo” function that avoids repeated flipping of the same bit. The two algorithms have comparable simplicity, but simulations show that the soft-decision algorithm results in both improved performance and—because fewer decoding iterations are necessary—improved speed.

Index Terms— Codes, decoding, error correction, iterative methods.

I. INTRODUCTION

SIPSER AND Spielman present simple and fast “bit-flipping” decoding procedures for expander codes [1], which are derived from bipartite expander graphs and can be classified as Gallager low-density parity-check codes [2]. In fact, Sipser and Spielman’s simple parallel decoding algorithm is equivalent to algorithms previously proposed by Gallager [2] and by Zyablov and Pinsker [3] for decoding low-density parity-check codes.

Sipser and Spielman’s decoding algorithms for their expander codes [1] are fast and simple but based on hard symbol decisions. In this paper, we present a soft-decision decoding algorithm for expander codes that maintains the simple elegance of Sipser and Spielman’s algorithm, yet improves performance both in terms of the decoded block error rate and in terms of computational load.

II. EXPANDER CODES

Sipser and Spielman’s expander codes [1] are derived from bipartite expander graphs. An expander graph is a graph in which every set of vertices expands by an unusually large factor, i.e., has an unusually large number of neighbors. A bipartite graph is a graph in which all vertices are partitioned into two mutually exclusive sets such that no edges exist between vertices in the same set. In particular, a bipartite graph is called (c, d) -regular if all the vertices in one set have degree c and all the vertices in the other have degree d .

Manuscript received May 16, 1997. The associate editor coordinating the review of this letter and approving it for publication was Prof. Y. Bar-Ness.

A. M. Chan was with the Department of Electrical and Computer Engineering, University of Toronto, Toronto, ON M5S 3G4, Canada. He is now with the Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge, MA 02139 USA.

F. R. Kschischang is with the Department of Electrical and Computer Engineering, University of Toronto, Toronto, ON M5S 3G4, Canada (e-mail: frank@comm.utoronto.ca).

Publisher Item Identifier S 1089-7798(98)05582-3.

Let B be a (c, d) -regular graph between a set of n vertices $\{v_1, \dots, v_n\}$ called variables and a set of (cn/d) vertices $\{C_1, \dots, C_{cn/d}\}$ called constraints, where $c < d$. Let $b(i, j)$ be a function such that, for each constraint C_i , the variables neighboring C_i are $v_{b(i,1)}, \dots, v_{b(i,d)}$. Let S be a binary block code of length d and rate $R > (c-1)/c$. Then the expander code $C(B, S)$ is the code of length n and rate at least $cR - (c-1)$ whose codewords are the words (x_1, \dots, x_n) such that $(x_{b(i,1)}, \dots, x_{b(i,d)})$ is a codeword of S for $1 \leq i \leq (cn/d)$. If B is a sufficiently good expander and if S is a sufficiently good code, then $C(B, S)$ will be a good code [1].

When S is a rate $(d-1)/d$ simple parity check (SPC) code, $C(B, S)$ is a Gallager low-density parity-check code [2]. Sipser and Spielman [1] observed the best performance using such expander codes; therefore we also focus our attention on these codes.

III. DECODING ALGORITHMS

When a block is transmitted over a noisy binary channel, some variables are received in error. A constraint C_i is said to be *satisfied* by a word (x_1, \dots, x_n) if $(x_{b(i,1)}, \dots, x_{b(i,d)})$ is a codeword of S ; otherwise, the constraint is *unsatisfied*. In the case that S is an SPC code, a constraint is satisfied if its neighboring variables have even parity. The decoder must somehow identify the erroneous variables and then flip them (i.e., complement their values) in order to satisfy all the constraints and thus recover the transmitted block.

The criterion for selecting variables to flip is quite simple. Whenever the decoder flips a variable, the unsatisfied constraints neighboring the variable become satisfied, while the satisfied constraints neighboring the variable become unsatisfied. Therefore, if the decoder flips a variable that appears in more unsatisfied than satisfied constraints, then the total number of unsatisfied constraints decreases. For a variable v , we denote by $U(v)$ the number of unsatisfied constraints in which v appears.

In this section, three decoding algorithms will be described. The first two algorithms are hard-decision algorithms similar to Sipser and Spielman’s algorithms [1], while the third is a soft-decision algorithm.

Parallel Decoding Algorithm:

- A. Identify the set of variables v for which $U(v)$ is largest.
- B. In parallel, flip each of those variables.
- C. Repeat these steps until $U(v) = 0$ for all v .

The parallel decoding algorithm has a natural sequential analog in which stepB is replaced with

Sequential Decoding Algorithm:

B. Randomly select one of those variables and flip it.

Unlike the algorithms presented in [1], these algorithms terminate when all constraints are satisfied, i.e., $U(v) = 0$ for all v . This feature allows the algorithms to continue and perhaps converge, even when no variable appears in more unsatisfied than satisfied constraints. Although flipping variables with more satisfied than unsatisfied constraints may not seem beneficial, the idea behind it is analogous to making negative progress in order to escape local minima in “slope-descent” algorithms. It is possible that some blocks may not converge at all; hence a maximum number of allowable iterations of the decoding loop must be set. Should this maximum be exceeded, the block is declared undecodable and the algorithm terminates.

Although both hard-decision algorithms have comparable performance and comparable complexity, the sequential algorithm is considerably faster in software than its parallel counterpart. Therefore, our soft-decision algorithm is an attempt to improve the performance of the hard-decision sequential algorithm.

We consider a binary system based on antipodal (± 1) signaling over a zero-mean white Gaussian noise channel with sample variance σ_N^2 . The channel output is denoted as y . We take $\text{sgn}(y)$ as a hard-decision of the corresponding transmitted bit, and $|y|$ as a measure of the reliability of this decision. Our soft-decision sequential decoding algorithm still requires hard decisions to be made prior to decoding, but it also takes into account the reliability of each individual decision during the decoding process.

In the soft-decision algorithm, we replace step B with

Soft Decision Sequential Decoding Algorithm:

B. Of these variables, flip the least reliable one. That variable cannot be flipped again until b iterations later, i.e., flipping that variable is “taboo” for the next b iterations.

Although the soft-decision algorithm is sequential in nature, there are two major differences between this algorithm and the hard-decision sequential algorithm.

Unlike the hard-decision algorithm, which randomly selects a variable v to flip from among those for which $U(v)$ is largest, the soft-decision algorithm selects the least reliable of these variables. This modification improves the likelihood of flipping erroneous variables.

However, now there is the increased probability of endlessly flipping the same unreliable variables over and over again, leading to a block that never converges to a valid codeword. To help avoid this, the soft-decision algorithm uses the concept of “taboos” [4]. Essentially, if a variable is flipped, it cannot be flipped again until b iterations later. We refer to b as the *taboo length*.

The algorithm can keep track of which variables are taboo by associating with each variable a taboo number. A variable can be flipped only if the current iteration number is equal to or less than the taboo number of that variable. Before a block is decoded, all variables are assigned a taboo number of 1, which means that any variable can be flipped during the first iteration. Any time a variable is flipped, its taboo number is set to b more than the current iteration number, preventing the variable from being flipped until b iterations later.

One way to implement the soft-decision algorithm is to maintain a collection of $c + 1$ queues, Q_0, \dots, Q_c , where Q_i contains variables v for which $U(v) = i$. We think of these queues as forming a hierarchy, with Q_c at the top, and Q_0 at the bottom. During each iteration, the algorithm chooses the least reliable nontaboo variable appearing in the highest nonempty $Q_i, i > 0$. If all variables appearing in $Q_i, i > 0$, are taboo, then the least reliable variable in the highest nonempty Q_i is chosen, i.e., the taboos are ignored. The chosen variable is flipped, the status of the c neighboring constraints is updated, and the cd variables adjacent to the updated constraints are re-queued. The algorithm terminates when all variables are in Q_0 or when the number of iterations exceeds the maximum allowed. In the former case, the values of the variables are output; in the latter case, the corresponding block is declared to be undecodable.

The main difference between this soft-decision algorithm and the sequential hard-decision algorithm of [1] is the method used to determine which variable to flip. In our implementation of the soft-decision algorithm, the variable to be flipped during each iteration is located by performing an $O(n)$ search. (This could be improved upon by maintaining sorted queues.) In practice, for moderate values of signal-to-noise ratio (SNR), we found that the size of the queues tapered toward the top of the queue hierarchy, making it possible to identify the bit to flip after searching through only a tiny fraction of n variables.

This search procedure makes the soft-decision algorithm slightly more complicated (per iteration) than its hard-decision counterpart. However, because the soft-decision algorithm does a better job of identifying which variable to flip, as described in Section IV, the number of iterations required by the soft-decision algorithm is significantly smaller than that required by the hard-decision algorithm, resulting in a net decrease in overall decoding time for practical block lengths.

IV. EXPERIMENTAL RESULTS

We implemented a program to test the performance of the decoding algorithms. The program obtains a (c, d) -regular bipartite graph by choosing a random matching between n c -regular vertices and (cn/d) d -regular vertices. As long as the random graph is large, it will have high expansion with high probability [1]. Without loss of generality, we assumed that the all-zero codeword was sent in all cases.

A. Optimum Length of Taboo

The relationship between block length n , taboo length b , and soft-decision decoding performance was examined using a $(4, 8)$ -regular graph. The noise variance σ_N^2 was set at 0.3721, while n and b were varied. For each set of parameters, the

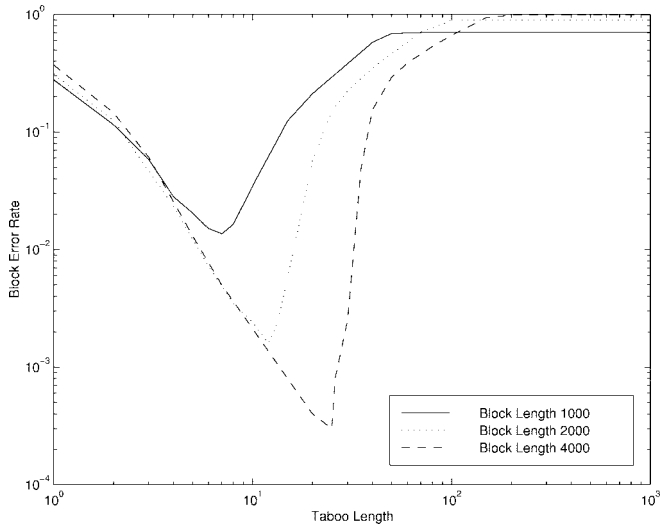


Fig. 1. Decoding performance versus taboo length for rate 1/2 expander codes, with $\sigma_N^2 = 0.3721$.

soft-decision algorithm attempted to correct 10 000 blocks. The results, presented in Fig. 1, indicate that for a given n and σ_N^2 , an optimum b exists. Intuitively, such a finding makes sense. When b is small, the same unreliable variables may be flipped over and over again, thus leading to an undecodable block. The special case $b = 1$ is equivalent to the case in which taboos do not exist. When b is large and the algorithm flips a variable that is *not* erroneous, the algorithm will have difficulty correcting this “mistake,” because it must wait until many iterations later before getting the opportunity to flip the variable back to its original correct value. Thus, a large b causes a degradation of performance.

The results also indicate that expander codes with larger n perform better (with a good choice of b), which is expected since large graphs tend to have greater expansion.

Most interestingly though, the results suggest that the optimum value of b is proportional to n . A similar relationship was observed using (5, 20)-regular graphs and values of n as high as 10 000, 20 000, and 40 000. The relationship can be explained by considering the following heuristic argument. Suppose the soft-decision algorithm desires to flip a particular variable v every so often for a given n . When n is doubled, the algorithm has twice as many other variables that it can try flipping. For optimum performance, the algorithm should now flip twice as many other variables before going back as a “last resort” to flip v . Therefore, the optimum value of b should be approximately proportional to n .

The relationship between σ_N^2 , b , and soft-decision decoding performance was also examined. In general, the optimum value of b decreases with decreasing σ_N^2 .

B. A Comparison of Decoding Algorithms

The hard-decision and soft-decision sequential decoding algorithms were compared using expander codes of length 4000. Each algorithm attempted to correct 10 000 blocks for a given code rate and σ_N^2 . A suitable value of b was chosen for the soft-decision algorithm. The results of this experiment are presented in Fig. 2, which plots the block error rate against E_b/N_o , defined as $1/(2R\sigma_N^2)$, where R is the expander code

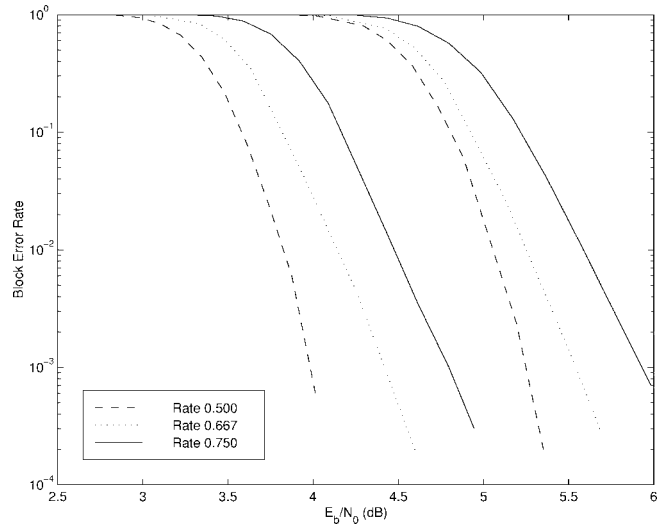


Fig. 2. Comparing soft-decision and hard-decision sequential decoding performance for expander codes of length 4000 in Gaussian noise. Two curves are plotted for each of the code rates indicated; the leftmost curve corresponds to the soft-decision algorithm in each case.

rate. For a given block error rate (on the order of 5×10^{-4}), the soft-decision algorithm is approximately 1.2 dB better than the hard-decision algorithm.

As mentioned in Section III, the actual number of variables searched through during each iteration of the soft-decision algorithm was observed to be only a small fraction of n . Furthermore, the algorithm usually chose the proper bits to flip, so iterations were not wasted in flipping bits that were already correct. Specifically, for the expander code rates and values of σ_N^2 simulated, the soft-decision algorithm “wasted” 50%–95% fewer iterations than the hard-decision sequential algorithm. Thus, a slight increase in the number of operations performed per iteration results in significantly fewer iterations being performed. The net effect is a decrease in overall decoding time for the soft-decision algorithm.

V. CONCLUSIONS

The soft-decision algorithm modifies the hard-decision sequential algorithm in two main ways. First, the soft-decision algorithm uses the sample value obtained at the receiver in each signaling interval as a measure of reliability—information that the hard-decision sequential algorithm ignores. Second, the soft-decision algorithm uses the concept of taboos. The combination of both these ideas results in improved performance and improved decoding speed, while still maintaining the simplicity inherent in Sipser and Spielman’s bit-flipping approach.

REFERENCES

- [1] M. Sipser and D. A. Spielman, “Expander codes,” *IEEE Trans. Inform. Theory*, vol. 42, pp. 1710–1722, Nov. 1996.
- [2] R. G. Gallager, *Low-Density Parity-Check Codes*. Cambridge, MA: MIT Press, 1963.
- [3] V. V. Zyablov and M. S. Pinsker, “Estimation of the error-correction complexity for Gallager low-density codes,” *Probl. Inform. Transm.*, vol. 11, no. 1, pp. 18–28, 1976.
- [4] F. Glover, E. Taillard, and D. de Warra, “A user’s guide to tabu search,” *Ann. Oper. Res.*, vol. 41, pp. 3–28, 1993.