On the Fly Gaussian Elimination for LT Codes

Valerio Bioglio, Marco Grangetto, *Senior Member, IEEE,* Rossano Gaeta, and Matteo Sereno, *Member, IEEE*

***Abstract*—We propose an improved algorithm for decoding LT codes using Gaussian Elimination. Our algorithm performs useful processing at each coded packet arrival thus distributing the decoding work during all packets reception, obtaining a shorter actual decoding time. Furthermore, using a swap heuristic the decoding matrix is kept sparse, decreasing the cost of both triangularization and back-substitution steps.**

***Index Terms*—LT codes, Gaussian elimination decoding, incremental decoding.**

I. Introduction and Background

RECENTLY, rateless codes have attracted much attention in the research community. Such codes, the most well known being LT [1] and Raptor [2] codes, are a class of erasure codes capable to reach optimal erasure recovery on the binary erasure channels (BEC) without fixing the rate. Belief Propagation (BP) algorithm presented in [1] is a fast decoding algorithm for LT codes, but for small code block length k, Gaussian Elimination (GE) (or Incremental Gaussian Elimination (IG) proposed in [3]) could be used. BP is simple and fast but for small k it requires a large overhead to decode while IG is slower but with a smaller overhead. Both algorithms execute almost all their operations at the end of the transmission, so their time complexity is concentrated on the last packets arrival. In this letter, we propose a GE-like algorithm, called On the Fly Gaussian Elimination (OFG), to decode LT codes. Our algorithm exhibits a noticeable computational saving w.r.t. IG and also spreads the computations over all packet arrivals. In this way we obtain a faster decoder due to the smaller overhead and more evenly distributed complexity.

II. Decoding LT Codes

In an LT code, k source packets $m = [m_0, m_1, \ldots, m_{k-1}]$ are encoded into a (theoretically) infinite stream of coded packets $y = [y_0, y_1, \ldots]$. A coded packet y_i is obtained by choosing a degree d, i.e., d random integers $r_0, \ldots, r_{d-1}$ in the interval $[0, k-1]$, and xoring the source packets at the corresponding positions, so that $y_i = m_{r_0} \oplus m_{r_1} \oplus \ldots \oplus m_{r_{d-1}}$. Degree d is chosen using the Robust Soliton distribution $\mu(c, \delta)$ [1]. For every coded packet y_i the decoder needs an equation $b_i = [b_{i,0}, b_{i,1}, \ldots, b_{i,k-1}]$ of the positions of packets xored to obtain y_i, i.e., $b_{i,j} = 1$ if m_j was used to calculate y_i, $b_{i,j} = 0$ otherwise. Let us assume that the decoder receives n coded packets ($y = [y_0, y_1, \ldots, y_{n-1}]$) and call B the $n \times k$-matrix of equations: m can be decoded if the system $Bm = y$ can be solved. We can also consider the $k \times k$-matrix G and the k-vector y' obtained from B and from y by deleting redundant, i.e., linear dependent, equations. The related subsystem $Gm = y'$ has to be solved to decode m. Of course, both systems yield the same solution. In the following the i-th row of a matrix Q will be denoted as $Q[i]$.

There are several algorithms used for solving such a system. The most used is the BP algorithm [1]. The system can also be solved by the classical GE algorithm or by its improved version IG [3].

BP algorithm: $B[i_0]$ that contains only one 1 in column j_0 (degree 1 packet) is selected; it follows that $m_{j_0} = y_{i_0}$, all the 1's in column j_0 are canceled and their y_i are xored with y_{i_0}. The above process is iterated until the matrix B becomes all-0 matrix (decoding success) or until no more degree 1 packets can be found (decoding failure). In case of failure a new packet is received, the 1's in the known positions of the corresponding equation are canceled and decoding is reattempted.

IG algorithm: The GE triangularization step is performed only once after the reception of k encoded packets $[y_0, \ldots, y_{k-1}]$. In case of GE failure (B is partially triangular) IG tries to fill the "bad" rows (rows without a 1 on the diagonal) using new coded packets and the corresponding equations. This process is incremental in the sense that rows of B and the new packets are xored and swapped without repeating an expensive GE step on the whole matrix. As soon as B turns into triangular form back substitution is used to complete decoding.

If n encoded packets are needed for decoding we say that the *overhead* of the code is $\epsilon = n/k - 1$. Clearly, $\epsilon \to 0$ for $k \to \infty$, but convergence speed depends also on the employed decoding algorithm. In Fig. 1 ϵ is shown versus k for GE and BP decoders for an LT code with $\delta = 0.01$ and $c = 0.01, 0.1$. It can be noted that for GE ϵ converges faster than BP; therefore, for small k GE-like algorithms need less encoded packets to decode. Moreover GE performance is far less sensitive to the parameters chosen for the degree distribution, making the code design simpler. On the other hand, if we compute the cost of an algorithm as the number of row xor and swap operations needed to decode then BP complexity turns out to be $O(k \log k)$ [1] while IG complexity is approximatively that of one single GE, i.e., $O(k^2)$ [3].

III. On the Fly Gaussian Elimination

On the Fly Gaussian Elimination (OFG) is a GE-like algorithm that does not wait for the first k packets to attempt the GE triangularization as in [3]; rather it builds a triangular matrix G by exploiting every received packet starting from the very first one. Moreover, OFG employs a swap heuristic that yields a sparse triangular matrix reducing the cost of

Manuscript received September 9, 2009. The associate editor coordinating the review of this letter and approving it for publication was V. Stankovic.

The authors are with the Dipartimento di Informatica, Università di Torino, Corso Svizzera 185, 10149 Torino, Italy (e-mail: {bioglio, grangetto, gaeta, sereno}@di.unito.it).

Digital Object Identifier 10.1109/LCOMM.2009.12.091824

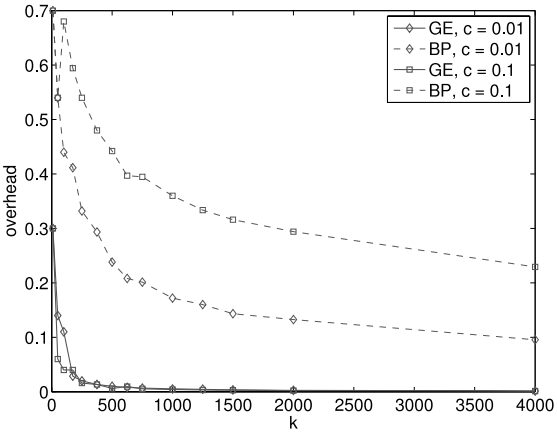

Fig. 1. Average overhead using $\mu(c, 0.01)$ distribution.

both row xor and swap operations as well as the final back substitution. The main idea is to write as soon as possible matrix G in a triangular form by deleting redundant equations on the fly. Informally, OFG works as follows (all the details are presented in Algorithm 1): assume G is a partially triangular $k \times k$-matrix, i.e., either a row has leftmost 1 on diagonal or it is all-zero. Upon receiving an encoded packet y_i with equation b_i we find the position of the leftmost 1 in b_i that we denote as s_i. If $G[s_i]$ is all-zero then we can replace $G[s_i]$ by b_i otherwise we xor b_i and $G[s_i]$ as well as y_i and y_{s_i}. We then obtain a new equation b'_i and a new coded packet y'_i. The new equation is such that the $s'_i > s_i$. The row finding and xoring are iterated until either the equation is placed into G or all 1's in the equation are canceled, i.e., the equation is discarded. The number of iterations depends on the probability to collide with a full row, that grows as G is being populated. We experimentally observed that keeping a sparser G markedly decreases the number of iterations per received packet, especially when G fills up. To this end we define the following swap heuristic that can be applied at any iteration: if $G[s]$ is full, i.e., it has its leftmost 1 on diagonal, but the equation to be inserted has a lower degree than $G[s]$, the equation and $G[s]$ are swapped along with the corresponding coded packets. Since the complexity to insert a row in G is $O(k)$, it turns out that the total complexity of OFG is $O(k^2)$. After the triangularization phase the source packets are computed by means of simple back-substitution (not included in Algorithm 1 for brevity).

We give an intuition of how OFG works by considering a simple example. G is the partially triangular matrix in Fig. 2(a) and $b_1 = [01100]$ is the new received equation. In this case $s_1 = 2$; since $G[2]$ is not full, b_1 inserted directly, thus obtaining G in Fig. 2(b). G is still not triangular ($G[5]$ is all zeros), therefore we need a new equation. Equation $b_2 = [11010]$ is received at this point. We get $s_2 = 1$, but $G[1]$ is already full; in this case b_2 and $G[1]$ are xored, obtaining $b'_2 = [01000]$. We should insert this equation in $G[2]$, but it is full. However, note that $G[2]$ has degree 2 while b'_2 has degree 1 and the swap occurs. We get $G[2] = [01000]$ and $b'_2 = [01100]$. Then, xoring $G[2]$ and b'_2 we obtain $b''_2 = [00100]$ that should go in $G[3]$. This latter is already full and its degree is greater than degree of b''_2, therefore we repeat the swap and xor operations. Finally, OFG leads to matrix G

$$\begin{matrix}1&0&0&1&0\\0&0&0&0&0\\0&0&1&1&0\\0&0&0&1&1\\0&0&0&0&0\end{matrix}\quad(a)\qquad\begin{matrix}1&0&0&1&0\\0&1&1&0&0\\0&0&1&1&0\\0&0&0&1&1\\0&0&0&0&0\end{matrix}\quad(b)$$

$$\begin{matrix}1&0&0&1&0\\0&1&0&0&0\\0&0&1&0&0\\0&0&0&1&0\\0&0&0&0&1\end{matrix}\quad(c)\qquad\begin{matrix}1&0&0&1&0\\0&1&1&0&0\\0&0&1&1&0\\0&0&0&1&1\\0&0&0&0&1\end{matrix}\quad(d)$$

Fig. 2. Triangularization step in the OFG algorithm.

as depicted in Fig. 2(c). Using OFG without the swap heuristic one obtains the matrix in Fig. 2(d). It is easily observed that the swap heuristic yields a sparser triangular matrix.

Algorithm 1 On the Fly Gaussian Elimination

```
Initialize k × k-matrix G, k-vectors Y and NumOnes to 0
Initialize EmptyRows = k
while EmptyRows > 0 do
  receive k-vector NewEq and encoded packet NewY
  s ← LeftmostOne (NewEq), EqOnes ← Degree(NewEq)
  while EqOnes > 0 and G[s][s] = 1 do
    if EqOnes ≥ NumOnes[s] then
      NewEq ← NewEq ⊕ G[s]
      NewY ← NewY ⊕ Y[s]
    else
      Swap(NewEq, G[s]), Swap(NewY, Y[s])
      NumOnes[s] ← EqOnes
    end if
    s ← LeftmostOne(NewEq)
    EqOnes ← Degree(NewEq)
  end while
  if EqOnes > 0 then
    G[s] ← NewEq and Y[s] ← NewY
    NumOnes[s] ← EqOnes
    EmptyRows ← EmptyRows − 1
  else
    delete NewEq
  end if
end while
```

IV. Simulation Results

We experimented BP, IG and OFG algorithms for several values of k using an LT code with $\delta = 0.01$ and $c = 0.01$. All the techniques have been implemented in C language using the same level of optimization in order to obtain fair comparisons. We run 1000 encodings/decodings for each algorithm and we present averages of the relevant performance indexes.

In Fig. 3 we show the complexity of the triangularization step, computed by counting the total number of row xor and swap operations, for OFG IG and BP vs. k. Moreover just for comparison, the cost of an ideal GE triangularization, i.e., performed only once as soon as B turns to be full rank, is reported. As shown in [3], the cost of IG equals that of a single GE triangularization, hence their curves in Fig. 3 are overlapped. Using OFG one halves the number of operations w.r.t. IG and GE while guaranteeing the same overhead. BP is still faster than OFG. Furthermore, Fig. 4 shows the number of 1's in G vs. k for OFG IG and GE. It can be noted that the proposed swap heuristic yields a sparser G in the OFG case.

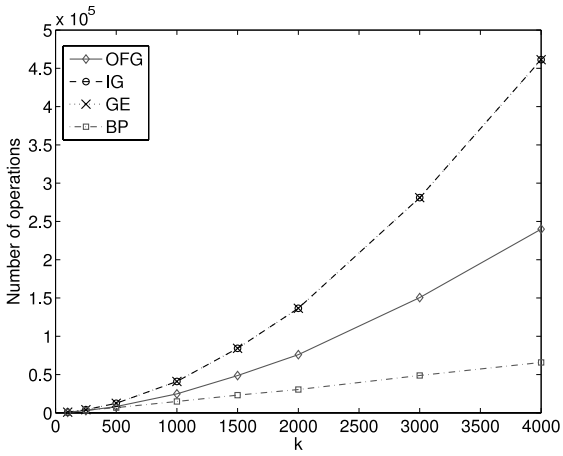

Fig. 3. Complexity of triangularization step.

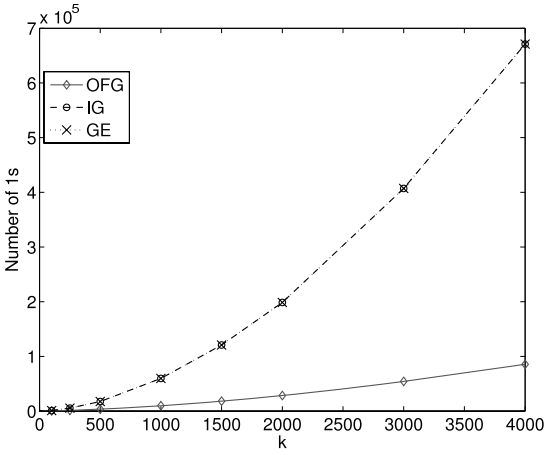

Fig. 4. Number of 1's in the triangular matrix (excluding diagonal).

Moreover, OFG computational effort is distributed on all packet receptions. This feature turns out to be of paramount importance when taking into account the packet reception delay. BP and IG spend almost all the computations after the required number of packets has been received and the actual decoding time is the sum of the time spent to receive the packets plus the time spent to solve the system.

On the contrary, OFG keeps triangulating the matrix while waiting for the next packets. In this way, OFG is able to decode almost immediately after the last packet arrival while BP and IG start only at this point most of their decoding operations. In Fig. 5 we report the normalized (over k) number of operations per packet as a function of the percentage of received packets. It can be noted that the maximum value in Fig. 5 is about 0.2, corresponding to only $k/5$ operations per packet. Clearly, the row insertion cost goes up with the number of received packets since the number of empty rows reduces. Finally, we measured the real decoding time with a multi-threading and concurrent implementation of the receiving and decoding tasks. The experiments have been worked out on a 2.66 GHz Intel Core Duo CPU equipped with 2 GB RAM. In Fig. 6 we show the CPU usage (%) as a function of the time, when the receiver downloads 1000 bytes coded packets at 1Mbps. LT coding with $k = 10000$ is used. We observe that OFG allows for a 12.5% reduction in the overall decoding time w.r.t. BP and a more remarkable 28.9% w.r.t. to IG. BP needs

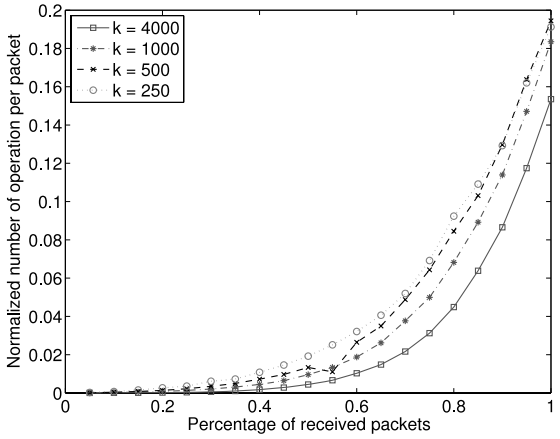

Fig. 5. Normalized operations per packet vs. percentage of received packets.

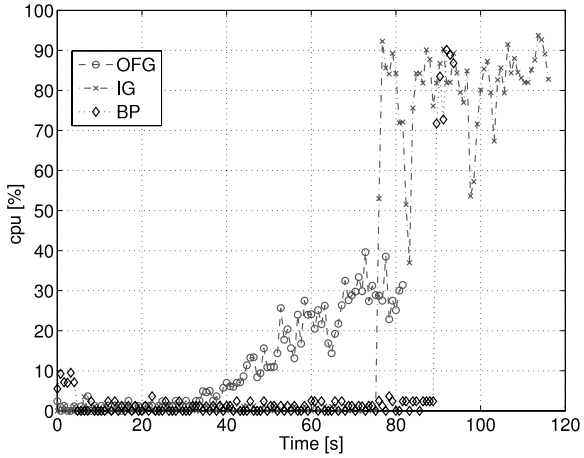

Fig. 6. CPU usage (%) as a function of the time for 1Mbps transmission.

more packets to start decoding due to the larger overhead ($\epsilon = 0.06$). On the other hand, OFG is able to decode almost immediately after the last packets arrival taking full advantage of the reduced overhead ($\epsilon = 5 \cdot 10^{-4}$) and exhibits a less intensive usage of the CPU. IG has a limited overhead as well, but it performs most of its operations at the end, turning out to be slowest solution in this scenario. In fact, the large peak for IG shows up at $T = 80$ s, which corresponds to the time needed to download the first k coded packets, and it is mainly due to the first and unique GE.

V. Conclusions

In this letter we proposed On the Fly Gaussian Elimination (OFG) that is an efficient version of Gaussian Elimination decoding for LT codes that spreads decoding complexity during packets reception. Future investigations include the analysis of the OFG performance for Raptor codes and the design of alternative degree distributions with desirable properties in terms of both overhead and decoding complexity.

References

[1] M. Luby, "LT codes," in *Proc. IEEE Symposium on Foundations of Computer Science*, Nov. 2002, pp. 271–280.

[2] A. Shokrollahi, "Raptor codes," *IEEE Trans. Inf. Theory*, vol. 52, no. 6, pp. 2551–2567, June 2006.

[3] S. Kim, K. Ko, and S. Chung, "Incremental Gaussian elimination decoding of Raptor codes over BEC," *IEEE Commun. Lett.*, vol. 12, no. 4, pp. 307–309, Apr. 2008.