Parallel versus sequential updating for Belief Propagation decoding

Haggai Kfir and Ido Kanter

Minerva Center and Department of Physics,

Bar-Ilan University, Ramat-Gan, 52900, Israel.

Abstract

A sequential updating scheme (SUS) for the belief propagation algorithm is proposed, and is compared with the parallel (regular) updating scheme (PUS). Simulation results on various codes indicate that the number of iterations of the belief algorithm for the SUS is about one half of the required iterations for the PUS, where both decoding algorithms have the same error correction properties. The complexity per iteration for both schemes is similar, resulting in a lower total complexity for the SUS. The explanation of this effect is related to the inter-iteration information sharing, which is a property of only the SUS, and which increases the "correction gain" per iteration.

PACS numbers: 89.70.+C 89.20.Kk

I. INTRODUCTION

Error correcting codes are essential part of modern communication, enabling reliable transmission over noisy channels. Bounds over the channels capacity were derived by Shannon in 1948 [1], but more than four decades passed before codes that nearly saturate the bound, such as Turbo code [2] and LDPC [3] were presented. In recent years, an interesting bridge was established between error correcting codes and statistical mechanics of disordered systems [4–6].

It is well known that as the noise level in a channel increases, the decoding time (measured in algorithm iterations) also increases.[7, section 3.2]. Furthermore, as the noise f approaches the threshold level, f_c, the number of iterations diverges as a power-law, $t \propto 1/(f_c - f)$[8]. Hence, the acceleration of the decoding process, or the reduction of the required number of iterations, is essential to ensure a smooth information flow when operating near the channels capacity.

In this paper we propose a variation of the well known Belief Propagation (BP) algorithm [9–11], which we label as "Sequential Updating Scheme" (SUS). The complexity per iteration of the SUS is similar to the complexity per iteration of the *regular* Belief Propagation (BP) algorithm, with Parallel Updating Scheme (PUS). However, simulations over a Binary Symmetric Channel (BSC) indicate that for a given code, the SUS requires about *one half* of the iterations in comparison to the PUS, while the averaged bit error probability, p_b, is the same.

This article is organized as follows: In section 2 the parallel and the sequential updating schemes are defined. The distribution of the decoding time obtained in simulations over BSC for both schemes are presented in section 3. In section 4 the complexities of the sequential and the parallel updating schemes are compared. A qualitative theoretical argument supporting the acceleration of the decoding procedure in the SUS is presented in section 5, and a detailed description of the simulations is presented in section 6. A brief conclusion is presented in section 7.

II. THE BP ALGORITHM

A. Notation

We focus on transmission over a BSC where each transmitted bit has a chance f to flip during transmission, and a chance $1-f$ of being transmitted correctly. Most of our results were obtained using Mackey and Neal's algorithm (MN) , described in detail in reference [9]. Briefly, the algorithm is defined as follows:

A word of size n is encoded into a codeword of size m, $(Rate = n/m)$, using the following binary matrices:

$A:$ a sparse matrix of dimensions $(m \times n)$

$B:$ a sparse and invertible matrix of dimensions $(m \times m)$.

The encoding of a word s into a codeword t is performed by:

$$t = B^{-1}As \,(mod\, 2) \tag{1}$$

During the transmission, a noise n is added to the data, and the received codeword r is:

$$r = t + n \,(mod\, 2) \tag{2}$$

The decoding is performed by calculating $z = B \cdot r$

$$z = B \cdot r = B \cdot (t+n) = B \cdot (B^{-1} \cdot A \cdot s + n) = A \cdot s + B \cdot n = [A, B][s, n], \tag{3}$$

where [ ] represents appending matrices or concatenating vectors.

Denoting $H = [A, B]$ and $x = [s, n]$, the decoding problem is to find the most probable x satisfying: $H \cdot x = z$ (mod2) , where:

H is $(m \times (n+m))$ matrix

z is the constraints (checks) vector of size m.

x is the unknown (variable) vector of size $n+m$, termed variable vector.

This problem can be solved by a BP algorithm. Following [9, 12], we refer to the elements of x as nodes on a graph represented by H, the elements of z being values of checks along the graph. The non-zero elements in a row i of H represent the bits of x participating in the corresponding check z_i. The non-zero elements in column j represent the checks in which the jth bit participates.

We follow Kanter and Saad's (KS) construction for the matrices A, B [8, 12, 13]. This construction is characterized by very sparse matrices and a cyclic form for B, together with relatively high error correction performance.

For each non-zero element in H, the algorithm calculates 4 values [10]. The coefficient q_{ij}^1 (q_{ij}^0) stands for the probability that the bit x_j is 1 (0), taking into account the information of all checks in which it participates, except for the ith check. The coefficient r_{ij}^1 (r_{ij}^0) indicates the probability that the bit x_j is 1 (0), taking into account the information of all bits participating in the ith check, except for the jth bit.

The algorithm is initialized as follows. The coefficient Q_j is set equal to our prior knowledge about that bit j. In our simulations we assume $Q_j = 0.5$ if it is a source bit ($j \leq N$), and $Q_j = f$ if it is a noise bit ($j > N$). Then the q values are set: $q_{ij}^1 = Q_j$; $q_{ij}^0 = 1 - q_{ij}^1$ for all non-zero elements in the jth column.

B. Parallel Updating Scheme (PUS)

The PUS consists of alternating horizontal and vertical passes over the H matrix. Each pair of horizontal and vertical passes is defined as an iteration. In the horizontal pass, all the r_{ij} coefficients are updated, row after row:

$$r_{ij}^0 = \sum_{(all\, configurations\, with\, x_j=0,\, satisfing\, z_i)} \prod_{j' \neq j} q_{ij'}^{x_{j'}} \tag{4}$$

$$r_{ij}^1 = \sum_{(all\, configurations\, with\, x_j=1,\, satisfing\, z_i)} \prod_{j' \neq j} q_{ij'}^{x_{j'}} \tag{5}$$

where it is clear that the multiplication is performed only over the non-zero elements of the matrix H.

A practical implementation of (4) and (5) is carried out by computing the differences $\delta q_{ij} \equiv q_{ij}^0 - q_{ij}^1$, and $\delta r_{ij} \equiv r_{ij}^0 - r_{ij}^1$ is then obtained from the identity:

$$\delta r_{ij} = (-1)^{z_i} \prod_{j' \neq j} \delta q_{ij'}. \tag{6}$$

From the normalization condition $r_{ij}^0 + r_{ij}^1 = 1$ one can find:

$$r_{ij}^0 = (1 + \delta r_{ij})/2 \; ; \; r_{ij}^1 = (1 - \delta r_{ij})/2 \tag{7}$$

(For a detailed description of this method, see [9]).

In the vertical pass, all q_{ij}^1, q_{ij}^0 are computed, column by column, using the updated values of r_{ij}^1, r_{ij}^0:

$$q_{ij}^0 = \alpha_{ij} p_j^0 \prod_{i' \neq i} r_{i'j}^0 \tag{8}$$

$$q_{ij}^1 = \alpha_{ij} p_j^1 \prod_{i' \neq i} r_{i'j}^1 \tag{9}$$

where α_{ij} is a normalization factor, chosen to satisfy $q_{ij}^0 + q_{ij}^1 = 1$, and p_j^0 , p_j^1 are the priors. Now the pseudo-posterior probability vector Q can be computed by:

$$Q_j^0 = \alpha_j p_j^0 \prod_i r_{ij}^0 \tag{10}$$

$$Q_j^1 = \alpha_j p_j^1 \prod_i r_{ij}^1 \tag{11}$$

Again, α_j is a normalization constant satisfying $Q_j^1 + Q_j^0 = 1$, and i runs only over non-zero elements of H. Each iteration ends with a clipping of Q to the variable vector x : if $Q_j > 0.5$ then $x_j = 1$, else: $x_j = 0$. At the end of each iteration a convergence test, checking if x solves $Hx = z$, is performed. If some of the m equations are violated, the algorithm turns to the next iteration until a pre-defined maximal number of iterations is reached with no convergence (our halting criteria are described in detail in section 6). Note that there is no inter-iteration information exchange between the bits: all r_{ij} values are updated using the previous iteration data.

C. Sequential Updating Scheme (SUS)

In the SUS, we perform the horizontal and vertical passes separately for each bit in x. A single sequential iteration for the bit x_j consists of the following steps:

1. For a given j all r_{ij} are updated. More precisely, for all non-zero elements in column j of H, use (6,7) for updating r_{ij}. Note that this is only a *partial* horizontal pass, since only r_{ij}'s belonging to a specific column are updated.

2. After all r_{ij} belonging to a column j are updated, a vertical pass as defined in (8,9) is performed over column j. Again, this is a *partial* vertical pass, referring only to one column.

3. Steps 1-2 are repeated for the next column , until all columns in H are updated.

4. Finally, the pseudo posterior probability value Q_j, is calculated by (10,11).

After all variable nodes are updated, the algorithm continues as for the parallel scheme: clipping, checking the validity of the m equations and proceeding to the next iteration.

III. RESULTS

We performed simulations of decoding over a BSC using various rates, block length and flip rates (f) and with the following constructions: (a) KS [8] construction; and (b) Irregular LDPC codes, following the Luby - Mitzenmacher - Shokrollahi - Spielman construction (LMSS), described in [3]. Note that the LMSS code differs slightly from the MN code, but still has a similar form of the BP scheme. We compare the distribution of the convergence times of the PUS and SUS by decoding of the same codewords (samples).

Figure 1 presents the distribution of the convergence times (measured in algorithm iterations) for PUS (filled bars) and SUS (empty bars) . The code rate is $1/2$, $f = 0.08$ (the channel capacity is ≈ 0.11) and the block length is $10,000$. The statistics were collected over at least $3,000$ different samples. The converging time for the SUS is about one half of the converging time for the PUS. The average convergence time for the PUS is 32.12 iterations for the KS construction; 28.52 for the LMSS construction; while for the SUS the average convergence time is 16.7 and 16.32 iterations, respectively. It is worth mentioning that the superior decoding time does not damage the error correcting property of the code. In both constructions the observed bit error rate, p_b, after PUS or SUS, is nearly the same (see Table 1 for details).

In Figure 2 the ratio between the converging times, (SUS/PUS) per sample is plotted (time is measured in algorithm iterations). For the vast majority of the samples this is very close to the average rate. This indicates that the double number of iterations for the PUS in comparison to the SUS is the typical result.

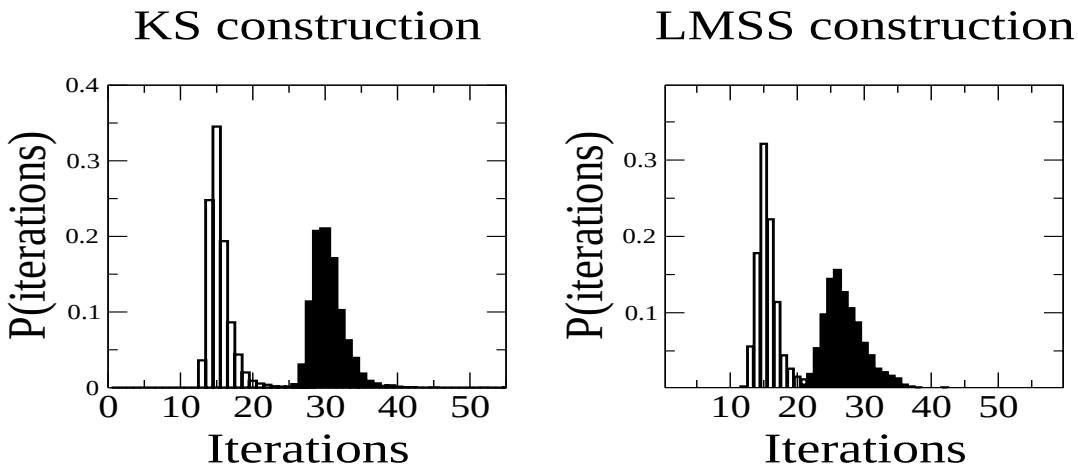

Figure 1: Distribution of the convergence times (measured in iterations) for PUS and SUS (filled and empty bars, respectively), for the KS and LMSS constructions. Rate is $1/2$, $f = 0.08$ and block size $n = 10,000$.

Table 1 presents similar measurements for other rates and noise levels. Results indicate the following general rule. Independent of the construction, the noise level and the rate, the convergence time of the PUS is around double the number of iterations required to achieve convergence in the SUS. (As our statistics was collected over ~ 3000 samples of block size 10^4, we do not report the exact value for $p_b \leq 10^{-5}$).

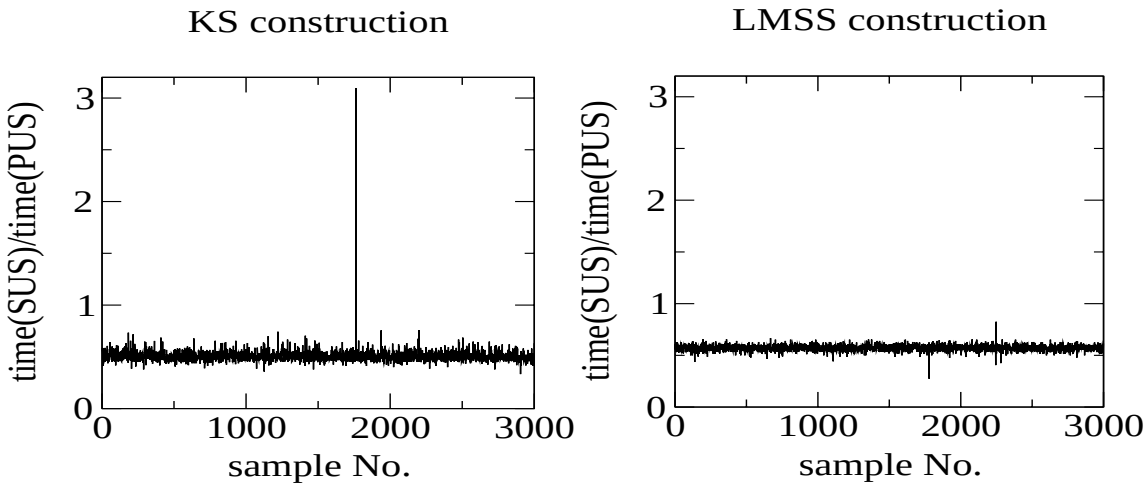

Figure 2: The ratio of convergence times, SUS time / PUS time, per sample (time is measured in algorithm iterations). The rate is almost a constant (0.5) independent of the particular sample.

construction	$< t_{PUS} >$ $(iterations)$	$< t_{SUS} >$ $(iterations)$	$\left\langle \frac{t_{SUS}}{t_{PUS}} \right\rangle$	$\frac{\langle t_{SUS} \rangle}{\langle t_{PUS} \rangle}$	bit error rate (PUS, SUS)
KS,R=1/2, f=0.09	58.3	28.4	0.488	0.504	$5.4 \cdot 10^{-5}$, $5.5 \cdot 10^{-5}$
KS, R=1/2, f=0.08	32.12	16.7	0.519	0.5076	$p_b \leq 10^{-5}, p_b \leq 10^{-5}$
KS, R=1/2, f=0.07	23.0	11.54	0.505	0.501	$p_b \leq 10^{-5}, p^b \leq 10^{-5}$
KS, R=1/3, f=0.159	65.58	32.54	0.496	0.5236	$3.4 \cdot 10^{-4}, 2.08 \cdot 10^{-4}$
KS,R=1/3, f=0.15	40.53	21.28	0.525	0.525	$p_b \leq 10^{-5}, p_b \leq 10^{-5}$
KS, R=1/5, f=0.23	103.9	54.66	0.526	0.5005	$1.9 \cdot 10^{-3}, 1.9 \cdot 10^{-3}$
KS, R=1/5, f=0.21	36.98	19.46	0.527	0.526	$p_b \leq 10^{-5}, p_b \leq 10^{-5}$
LMSS, R=1/2, f=0.08	28.52	16.32	0.572	0.573	$p_b \leq 10^{-5}, p_b \leq 10^{-5}$
LMSS,R=1/2, f=0.07	17.90	10.94	0.611	0.612	$p_b \leq 10^{-5}, p_b \leq 10^{-5}$

IV. COMPLEXITY PER ITERATION

In this section we show that the complexity per single iteration is almost the same for both methods. Hence, the gain in iterations represents the gain in decoding complexity.

For both schemes, the calculation of the pseudo posterior probabilities, the clipping and the convergence tests are identical, so they can be excluded from our discussion. Furthermore, the vertical passes in the two schemes are identical, hence the only remaining source for a possible difference in the complexity for the two schemes is the horizontal pass. For simplicity, we assume a regular matrix H of dimensions m rows by n columns, which has k nonzero elements per row and c nonzero elements per column. (One can easily extend the discussion to include irregular matrices, but the conclusions are the same).

In the PUS, each horizontal pass consists of k subtraction operations to find δq_{ij}'s and $(k-1)$ multiplications to find δr_{ij}'s (using (6)) for each bit. The calculation of r_{ij}^1 & r_{ij}^0 from δr_{ij} requires two additions and two multiplications (7). The total number of operations per iteration for the PUS is given by

additions:

$$m(k+2k) = 3mk \tag{12}$$

multiplications:

$$m(k(k-1)+2k) = mk(k+1) \tag{13}$$

In the SUS, horizontal passes are done separately for each bit, summing to $n \cdot c$ passes in total. Each horizontal pass consists of $k-1$ subtractions to find $\delta q_{ij'}$ for all participants in the check, except for the current bit, and $k-1$ multiplications are required to calculate δr_{ij}. The calculation of r_{ij}^1 & r_{ij}^0 from δr_{ij} in this scheme requires two additions and two multiplications per bit. Hence the total complexity is given by

additions:

$$nc(k-1+2) = nc(k+1) \tag{14}$$

multiplications:

$$nc(k-1+2) = nc(k+1) \tag{15}$$

Recalling that $mk = nc$, we have:

- (13) equals (15), so the same number of multiplications is performed in both cases.
- (14) becomes $m(k^2+k)$, which for $k > 2$ is larger than (12). However, for small k ($k \leq 5$ for KS, and $\langle k \rangle$ is of the same order for LMSS), the increment in the total fraction of additions is small. Furthermore, one must remember that *the complexity is dominated by the number of multiplications.*

Note that the abovementioned comparison was made under a straightforward implementation of both algorithms. In advanced algorithms the following improvements can be adopted in order to reduce the complexity of the schemes.

1. Some savings can be made for the horizontal passes in the PUS, for instance, computing $\prod \delta q_{ij}$ for the entire row, and dividing by each δq_{ij} element, or recomputing δq_{ij} only for updated bits in the SUS.
2. PUS can be implemented simultaneously over all checks (variables) using several processors. The implementation of SUS in parallel over a finite fraction of the checks (variables) is possible, but may require a special design.
3. SUS has some advantage in memory requirement, since only a column vector of the currently updated r_{ij} is required, whereas for the PUS the whole r_{ij} matrix must be retained simultaneously.

V. QUALITATIVE THEORETICAL EXPLANATION

The key difference between the two algorithms is the inter iteration information exchange, which is a property of the SUS only. Let us denote by $r_{ij}^t, q_{ij}^t \equiv$ the values computed in iteration t. In the PUS all r_{ij}^t values are determined by the q_{ij}^{t-1} values (values of the previous iteration), and the q_{ij}^t's are determined by these r_{ij}^t's . In the SUS, after a bit is updated, the following bits that share a check with it are already exposed to the updated information. For instance, assume x_j and x_k share a check i, i.e. $H_{ij} = H_{ik} = 1$, and assume $j < k$. In iteration t, r_{ij}^t is updated using q_{ik}^{t-1}; however, proceeding to the kth column, r_{ik}^t is updated using q_{ij}^t, the most recent available information.

In other words, in the SUS, the first bits to be updated utilize the previous iteration data (q_{ij}^{t-1}). A group of bits use mixed data from previous and current iterations and, finally, some of the bits are entirely updated by information from the current step (q_{ij}^{t}).

The gain in the number of iterations for the SUS can be qualitatively well understood by the following argument: Since the decoding procedure terminates successfully (with some small p_b) and the number of correct bits increases monotonically, it is evident that on average, the current knowledge is superior to the knowledge of previous iteration.[14]

An important question is raised regarding which part of the SUS is accelerated in comparison to the PUS. The acceleration of the SUS may be a result of one of the following regimes: (a) a faster asymptotic convergence; (b) a faster arrangement in the initial stage of the decoding from random initial conditions; or (c) a uniform acceleration over all the stages of the decoding.

In order to answer this question we perform the following advanced simulations. We run the PUS and record the number of correct bits in each iteration. The correction gain of each iteration is defined as the increment in the fraction of correct bits. At each step of the PUS we prepare another replica of the system with the same initial conditions, the same q_{ij} and r_{ij}, and run one iteration of the SUS. The correction gain of the SUS is then compared to that of the PUS. In Figure 3 we plot the rate between the sequential and parallel correction gains as a function of time (marked $\times$). This rate is nearly 2, with relatively small fluctuations along the decoding process. In other words, on the average the SUS corrects twice the number of bits in comparison to the PUS, independent of the state of the decoder. (The simulation was performed on the KS construction with rate 1/3, $f = 0.155$, block size 10000, 20 different samples, and all convergence times were normalized to a 0-1 scale.)

The observation that the correction gain uniformly distributed over all the stages of the decoding raises the question of whether there is a superior updating order of the bits producing a correction gain greater than 2. For one iteration of the KS construction, it may be better to update the bits from left to right than in the reverse order. For rate 1/3, for instance, the right-most part of the matrix (about 25% of the columns) contain only one non-zero element per column, and this element is also the last element in its check. These bits are entirely updated by current iteration data, resulting in an increased correction gain. At the left-most end, on the other hand, there are 7 non-zero elements per column (for rate 1/3 construction), so that only a small fraction of them are the "last bit" for all the checks

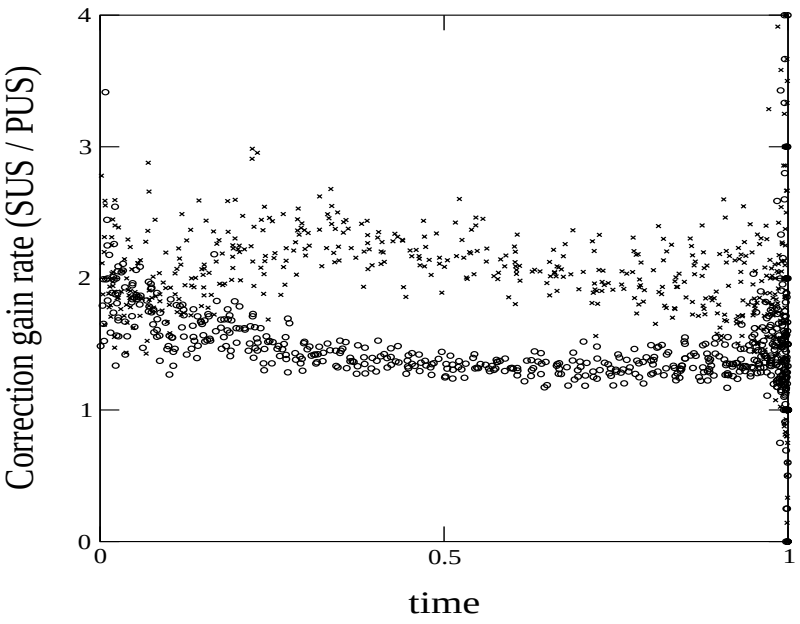

Figure 3: The rate between the correction gain of SUS and PUS versus the time (iterations). The data was collected over each iteration of a KS construction, rate $1/3$, $f = 0.155$ and block size $10,000$. The symbol "$\times$" represents a forward updating order, and "$\circ$" represents a reverse updating order. The correction gain rate is almost time independent, and is higher for forward updating due to the properties of the KS construction.

in which they participate. Most of the bits are updated by mixed information from the current and previous iteration, resulting in a smaller correction gain. In Figure 3 the rate between the correction gain for SUS and PUS for reversed (right to left) updating order is marked "$\circ$". This rate is evidently less than for the left to right updating. Preliminary simulations indicate that by carefully selecting the updating order one can save 10%-30% of the iterations relative to a plain left to right sequential updating.

VI. SIMULATIONS

In this sections the technical details of our simulations are described. We generate the H matrix at random, distributing the non-zero elements as evenly as possible without violating the constraint of the number of elements per row/column. No special attempt was made

to select a "good performing" matrix. For the KS structure, we generate the x vector as follows: The source bits were set to 1 or 0 with probability 0.5. The noise bits were set to 0, and then exactly a fraction f of the bits were selected randomly and flipped (f is the flip probability). The check vector z was computed by $z = Hx$, and the algorithm solved $Hx' = z$. We found p_b by comparing x & x', for the source region only. The source length selected was $n = 10,000$ (resulting in x of length $40,000$, and z of length $30,000$ for rate $1/3$).

For the LMSS structure, following [3] we always decoded the all-zero codeword, generating the noise vector n in the same way as described above. The check vector z was computed, $z = Hn$, and the algorithm solved $Hn' = z$. We found p_b by comparing n & n' (in the LMSS version the "decoding" ends when the noise vector is found and the transmitted vector, t, is related to the received vector, r by $t = r + n(\mathrm{mod}\ 2)$. Finding the source message from t is not defined as part of the decoding problem). We used a noise vector of length $20,000$, corresponding to a check vector of size $10,000$ (rate $1/2$).

In both cases the flip rate, f, was selected as being close enough to the critical rate for this block length such that the decoding is characterized by relatively long convergence times. However, the flip rate f was chosen not too close to the threshold in order to avoid a large fraction of non-converging samples. After the check vector z was constructed, it was decoded both in parallel and sequential schemes, and the number of iterations was monitored. We defined 3 halting criteria for the iterative process:

1. The outcome x' fully solves $Hx' = z$.

2. The algorithm reached a stationary state, namely, x' did not change over the last 10 iterations.

3. A predefined number of iteration was exceeded ("non-convergence"). This number was selected so as to be far larger than the average converging time (500 iterations in our case).

The vast majority of samples converged successfully. More precisely, less than 0.2% samples failed to converge or reach a non-solving stationary state.

VII. CONCLUSIONS

We demonstrated that the SUS outperforms the PUS in the convergence time aspect by a factor of about 2. Since the complexity per iteration of the two schemes is nearly the same, the gain in iterations is similar to the gain in the decoding complexity. The time gain is probably related to the inter-iteration information exchange, which is a property of the SUS. This explanation is also consistent with the observation that the gain is uniformly distributed over all the decoding stages. The question of whether the number of iterations can be reduced by a factor greater than 2 by updating the bits in a special order is currently under investigation.

We acknowledge fruitful discussions with D. Ben-Eli and I. Sutskover.

---

[1] C.E. Shannon, Bell Syst. Tech. j. **27**, 379(1948); **27**, 623 (1948).

[2] C. Berrou, A. Galvieux, and P. Thitimajshima, Proc. ICC '93 Geneve, Switzerland, pp. 1064, May 1993.

[3] M. G. Luby, M. Mitzenmacher, M. A. Shokrollahi and D. A. Spielman, *IEEE Transactions on Information Theory* **47(2)**, 585 (2001).

[4] N. Sourlas, Nature **339**, 693 (1989).

[5] A. Montanari and N. Sourlas, Europhys. J. B **18**, 107 (2000).

[6] Y. Kabashiba and D. Saad Europhys. Lett **45** (1), 97 (1999).

[7] T. Richardson, A. Shokrollahi and R. Urbanke, **Design of provably good low-density parity-check codes**, submitted to *IEEE Transactions on Information Theory* (1999).

[8] I. Kanter and D. Saad, *Phys. Rev. Lett.* **83**, 2660 (1999).

[9] David J. C. Mackey, *IEEE Transactions on Information Theory* **45**, 399 (1999).

[10] D. J. C. Mackey and R. M. Neal, *Electronics Letters* **33**, 457 (1997).

[11] R. G. Gallager, **Low Density Parity Check Codes**. *Research Monograph Series Vol. 21* (MIT, Cambridge, MA, 1963).

[12] I. Kanter and D. Saad, *Phys. Rev. E* **61,** 2137 (2000).

[13] I. Kanter and D. Saad, J Phys A: Math. Gen. **33**, 1675 (2000).

[14] Our simulations indicate that the number of correct bits versus time (iterations) is a *concave*

function (except for the last few iterations). This result leads to a faster convergence for the SUS (many-small-steps converging strategy) in comparison to the PUS.