Encoding information into constant weight words [1]

Nicolas Sendrier
INRIA Rocquencourt, B.P. 105, 78153 Le Chesnay Cedex, FRANCE
Email: Nicolas.Sendrier@inria.fr

***Abstract*— We present here a new algorithm for encoding binary information into words of prescribed length and weight. Existing solutions use a combinatorial approach and, though they are optimal in terms of information theory, they have a rather high algorithmic complexity as they require the computation of binomial coefficients. The solution we propose has linear complexity. The price to pay is variable length encoding and a small loss of (information theoretic) efficiency.**

I. Introduction

We address here the problem of encoding information, presumably produced by a uniform memoryless binary source, into binary words of prescribed length and Hamming weight. This is of interest, in particular, when one wishes to efficiently implement syndrome-based cryptosystems [Nie86], [Sid94]. In those systems, the cryptogram is the syndrome of a word of given weight and length, and thus information must be formatted into one or several such words. Moreover, since Niederreiter's encryption scheme is extremely fast (see [Sen02b]), this formatting process should be very fast too, preferably with a complexity depending linearly of the cryptogram size.

We will denote $W_{n,t}$ the set of binary words of length n and Hamming weight t. Our purpose is to exhibit a function which takes as input any binary sequence, and which outputs a sequence of words in $W_{n,t}$.

We consider the source $W_{n,t}$ equipped with a uniform distribution. An (optimal) binary encoder of that source will produce binary sequences very similar to those produced by a memoryless uniform binary source. If this encoder is fully decodable (see §II-A) the corresponding decoder will be an answer to our problem.

The exact solution is combinatorial and consists in indexing the elements of $W_{n,t}$ with the integers of the interval $[0, \binom{n}{t}[$. It was presented for instance in [Cov73]. Though it is optimal as a source coding algorithm, its implementation requires the computation of several binomial coefficient and thus has a quadratic complexity in the bloc length (that is $\log_2 \binom{n}{t}$). A first solution with a linear complexity, using Huffman codes, was proposed in [Sen95]. Its efficiency isn't very good, the distribution of the words in $W_{n,t}$ is flawed, and implementation was not satisfactory (a rather large Huffman tree had to be stored or computed). The proposal we make is based on run-length encoding [Gol66]. It is fairly easy to implement and runs in linear time. Though it isn't optimal, its efficiency is very close to 1.

[1] Work partly supported by the French Ministry of Research RNRT project X-Crypt

We will first give a quick review of the previous works of Cover on enumerative source encoding and of Golomb on run-length encoding. We then show how an adaptative version of the latter can provide an efficient solution to our problem. We also give simulation results of the various encoding methods in an implementation of Niederreiter encryption scheme.

II. Preliminaries and related works

A. Fully decodable codes

For any alphabet A, we denote by $A^* = \bigcup_{i\geq 0} A^i$ the set of all finite sequences of letters of A.

We consider a memoryless source of alphabet A, a code for that source is a mapping $f : A \to \{0,1\}^*$, its extension $F : A^* \to \{0,1\}^*$ is defined by $F(a_1, \ldots, a_\ell) = f(a_1) \parallel \ldots \parallel f(a_\ell)$, where $\parallel$ denotes the concatenation. The code f is uniquely decodable if F is injective. If in addition any binary sequence is the beginning of an encoded sequence (an element of $F(A^*)$), we will say that f is *fully decodable*.

If f is fully decodable, it is possible to define a mapping $G : \{0,1\}^* \to A^*$ such that, for all $(a_1, \ldots, a_\ell) = G(y)$ we have

- $F(a_1, \ldots, a_{\ell-1})$ is a strict prefix of y,
- y is a prefix of $F(a_1, \ldots, a_\ell)$.

The mapping G can be used to encode information bits into letters of A. The last letter a_ℓ is not uniquely defined. However, if its length is known, the sequence y is uniquely determined by $G(y)$. The efficiency is measured by comparing the length of y to the entropy of A^ℓ, that is, asymptotically, the efficiency of the code f.

B. Run-length encoding

We refer here to a source coding technique used for binary memoryless sources with a (highly) unbalanced probability law [Gol66]. We consider such a source and we assume $p = Prob(0) \geq 1/2$. For any non-negative integer d, we define the mappings $f_d : [0, d[\to \{0,1\}^*$ and $F_d : \mathbf{N} \to \{0,1\}^*$ as follows:

$$f_d(i) = \begin{cases} \text{base2}(i, u-1) & \text{if } 0 \leq i < 2^u - d \\ \text{base2}(i + 2^u - d, u) & \text{if } 2^u - d \leq i < d \end{cases}$$

$$F_d(i) = \begin{cases} 0 \parallel f_d(i) & \text{if } i < d \\ 1 \parallel F_d(i-d) & \text{else} \end{cases}$$

where u is the (only) integer such that $2^{u-1} < d \leq 2^u$, $\parallel$ denotes the concatenation, and $\text{base2}(x, l)$ denotes the l least significant bits of the integer x written in base 2, the most significant bit coming first. The source sequence

$0^{\delta_1}10^{\delta_2}10\ldots010^{\delta_s}$ will be encoded by $F_d(\delta_1) \parallel F_d(\delta_2) \parallel \ldots \parallel F_d(\delta_s)$. This procedure was described by Golomb [Gol66]. He also mentioned that the best choice for d was such that $(1-p)^d \approx 1/2$, that is $d \approx -1/log_2(1-p)$. In fact $(1-p)^d$ is the probability for any string of zeroes to have length $\geq d$. Encoding and decoding with f_d can be described as follows:

```
function encodefd
input: two integers δ and d
output: a binary string
    u ← ⌈log2(d)⌉
    if δ < 2^u − d
        u ← u − 1
    else
        δ ← δ + 2^u − d
    return base_2(δ, u)

function decodefd
input: an integer d, a binary stream B
output: an integer
    u ← ⌈log2(d)⌉
    δ ← read(B, u − 1)
    if δ ≥ 2^u − d
        δ ← 2δ + read(B, 1) − 2^u + d
    return δ
```

where $\texttt{base\_2}(\delta, u)$ returns a string composed of the u least significant bits of the binary decomposition of δ, most significant bit first, and $\texttt{read}(B, u)$ moves forward and reads u bits in the stream B and returns the integer whose binary decomposition has been read, most significant bit first.

C. Enumerative source encoding

Among other things, Cover's paper on enumerative source encoding [Cov73] presents a solution to our problem. It consists in indexing the set $W_{n,t}$ with the following mapping:

$$\begin{array}{rccc} \varphi & : & W_{n,t} & \rightarrow & [0, \binom{n}{t}[\\ & & (i_1, \ldots, i_t) & \mapsto & \binom{i_t-1}{t} + \ldots + \binom{i_1-1}{1} \end{array}$$

where $0 \leq i_1 < \ldots < i_t < n$ are the coordinates of the '1's. We can then write these indexes with $l = \lceil \log_2 \binom{n}{t} \rceil$ bits, or slightly less, in average, by using the mapping f_d with $d = \binom{n}{t}$.

The cost for computing φ is *a priori* quadratic in l. In fact we need to compute large binomial coefficients. It is possible to reduce the asymptotic complexity by using binary splitting techniques (a method that can be traced back to [Bre76a], [Bre76b], [BB87]) and either Karatsuba ($O(l^{\log_2 3})$) or fast Fourier transform ($O(l \log l)$) for the integer arithmetic. However, this would be difficult to implement, would be advantageous only for large values of the parameters, and still would not achieve linear time complexity. On the the other hand, precomputing the binomial coefficients would speed up the computation, but would require a memory quadratic in l.

Inverting φ efficiently is a bit trickier. Given an integer $0 \leq x < \binom{n}{t}$, we wish to compute a t-tuple such that $\varphi(i_1, \ldots, i_t) = x$. The integer i_t is the only one such that $\binom{i_t-1}{t} \leq x < \binom{i_t}{t}$. We can recover i_t by using the following inversion formula [Sen02a]

$$i_t = X + \frac{t-1}{2} + \frac{t^2-1}{24}\frac{1}{X} + O\left(\frac{1}{X^3}\right), X = (t!x)^{1/t}. \quad (1)$$

Repeated t times this will give us the all t-tuple. The fact that i_t is an integer makes the evaluation of (1) negligible in practice. The cost for computing $\varphi^{-1}(x)$ will be dominated by the cost for computing the t binomial coefficients $\binom{i_j-1}{j}$, $1 \leq j \leq t$.

D. Implementation of the Niederreiter encryption scheme

In the Niederreiter public key cryptosystem [Nie86], the public key is a binary parity check matrix H of size $r \times n$ and a integer t. In all this work, the secret code is a t-error correcting binary Goppa code of full length $n = 2^m$ and codimension $r = tm$. The encryption roughly consists in repeating the following steps until the source file is exhausted:

1) Read some bits in the source file $\rightarrow x$
2) Compute a constant weight word $y = \varphi^{-1}(x) \in W_{n,t}$
3) Compute the syndrome $z = yH^T$
4) Write z in the target file

This was implemented in C language on a PC equipped with a 2.4 Ghz Pentium 4 processor and running a Linux operating system. Binomial coefficients were computed with the GMP[2] package. The encryption rate we measured to encrypt a single 100 Mbytes file are given in table I. Most of the CPU time was used by the source encoding part (step 2). For instance,

n	t	$\log_2 \binom{n}{t}$	information rate
1024	50	284.0	2.0 Mbits/s
2048	29	215.9	4.1 Mbits/s
4096	9	89.5	9.5 Mbits/s
65536	9	125.5	10.6 Mbits/s

TABLE I
ENCRYPTION RATE FOR NIEDERREITER'S SYSTEM

for $(n, t) = (2048, 29)$ the encryption of the 100 Mbytes file takes 194 seconds, almost 180 seconds are used to translate information bits into words of $W_{2048,29}$, while the syndrome computation requires only 5 seconds (the remaining time is for inputs and outputs). In that particular case, the source coding is about 35 times more expensive than the syndrome computation. The purpose of the paper is to substitute to step 2 a source coding algorithm that will have a running time comparable with step 3.

III. SOURCE CODING OF CONSTANT WEIGHT WORDS

If we state our problem in a source coding context, our goal is to find an optimal binary code for $W_{n,t}$ equipped with the uniform distribution. If we manage to simplify the source

[2]GMP stands for GNU Multiple Precision Arithmetic Library, `http://www.swox.com/gmp/`

model, we might be able to find a fast encoder, and hopefully loose little in efficiency.

A very crude model, would be a binary memoryless source with probabilities $P(0) = 1 - t/n$ and $P(1) = t/n$ to which we could apply run-length coding. This approach has two flaws, first it does not take into accounts the correlations, and, more problematic, it does not lead to a fully decodable code of $W_{n,t}$. Encoding the run-length with a Huffman code using the probability distribution (2) is more efficient [Sen95] but does not completely resolve those flaws. Our new design will use Golomb's run-length coding but the parameters will be constantly adjusted. The result is an efficient fully decodable code.

A. Statistics

We will represent an element of $W_{n,t}$ as a t-tuple of integers $(\delta_1, \ldots, \delta_t)$, which are the lengths of the longest strings of consecutive '0's. If $i_1 < \ldots < i_t$ are the coordinates of the '1's in increasing order, we have $\delta_1 = i_1 - 1$, $\delta_2 = i_2 - i_1 - 1$, $\ldots$, $\delta_t = i_t - i_{t-1} - 1$.

The probability distributions of δ_j, $1 \leq j \leq t$ viewed as a random variable is independent of j. We have

$$Prob(\delta_j = s) = P_{n,t}(s) = \frac{\binom{n-s-1}{t-1}}{\binom{n}{t}}. \tag{2}$$

The random variables δ_j are not mutually independent though. We have

$$Prob(\delta_{j+1} = s \mid \delta_1, \ldots, \delta_j) = P_{n',t-j}(s), \tag{3}$$

with $n' = n - \delta_1 - \ldots - \delta_j - j$.

B. Encoding the first string of zeroes

Let $(\delta_1, \ldots, \delta_t)$ be a word of $W_{n,t}$, we wish to efficiently encode δ_1, that is the alphabet $[0, n-t]$ equipped with the probability law $Prob(\delta_1 = s)$, $0 \leq s \leq n - t$, given in (2). Using a Huffman or arithmetic code for this source would require either too much computation either too much memory. However, if we denote d the smallest integer such that $\sum_{s \geq d} P_{n,t}(s) < 1/2$, we have a situation that is very similar to run-length encoding, and we may try to use f_d to encode $\delta_1 < d$ and use an additional bit for the case $\delta_1 \geq d$. This leads to the following recursive encoder:

$$\varphi_{n,t}(\delta_1) = \begin{cases} 0 \parallel f_d(\delta_1) & \text{if } \delta_1 < d \\ 1 \parallel \varphi_{n-d,t}(\delta_1 - d) & \text{else} \end{cases}$$

This strategy will be locally optimal. The idea is now to change the value of d at each recursive call.

1) Optimal value of d: We have $\sum_{s \geq d} P_{n,t}(s) = \binom{n-d}{t} / \binom{n}{t}$, and thus the value of d we are looking for will verify $\binom{n}{t} \approx 2\binom{n-d}{t}$, that is (using formula (1))

$$d \approx \left(n - \frac{t-1}{2}\right)\left(1 - \frac{1}{2^{1/t}}\right) \approx \frac{\ln(2)}{t}\left(n - \frac{t-1}{2}\right). \tag{4}$$

The rightmost approximation is valid only for large t.

C. Encoding the whole word

We assume that we have for all n and t an encoder $\varphi_{n,t}$ for the source $[0, n-t]$ equipped with the probability law $P_{n,t}$. From that family of encoder, we can recursively define an encoder for $W_{n,t}$:

$$\Psi_{n,t}(\delta_1, \delta_2, \ldots, \delta_t) = \varphi_{n,t}(\delta_1) \parallel \Psi_{n-1-\delta_1,t-1}(\delta_2, \ldots, \delta_t) \tag{5}$$

From the conditional probabilities given in (3), this is enough to take into account the correlations between the various δ_j. Given the value of δ_1, the statistics of the $n - \delta_1 - 1$ remaining bits are exactly those of the set $W_{n-1-\delta_1,t-1}$ equipped with a uniform distribution.

The two following functions `CWtoB` (*constant-weight to binary*) and `BtoCW` (*binary to constant-weight*) implement respectively $\Psi_{n,t}$ and its inverse. The third argument of `BtoCW` has a value of 0 at the first call.

```
function CWtoB
input: two integers n and t, a t-tuple (δ_1, δ_2, ..., δ_t)
output: a binary string
    if t = 0 or n ≤ t
        return
    d ← best_d(n, t)
    if δ_1 ≥ d
        return 1 ∥ CWtoB(n − d, t, (δ_1 − d, δ_2, ..., δ_t))
    else
        s ← 0 ∥ encodefd(δ_1, d)
        return s ∥ CWtoB(n − δ_1 − 1, t − 1, (δ_2, ..., δ_t))
```

```
function BtoCW
input: three integers n, t and δ, a binary stream B
output: a t-tuple of integers
    if t = 0
        return
    else if n ≤ t
        return δ, BtoCW(n − 1, t − 1, 0, B)
    else
        d ← best_d(n, t)
        if read(B, 1) = 1
            return BtoCW(n − d, t, δ + d, B)
        else
            i ← decodefd(d, B)
            return δ + i, BtoCW(n − i − 1, t − 1, 0, B)
```

where `best_d`(n, t) is an integer such that $1 \leq$ `best_d`$(n, t) \leq n - t$. It is best to choose it close to the number defined by (4). In fact it can take any value in the range, the functions will still terminate and be the converse of each other, but if we are too far from (4) the efficiency the reduced.

D. Encoding information into constant weight words

Proving directly that $\Psi_{n,t}$ is fully decodable seems difficult. Instead we can argue, which is the same, that given sufficient input bits, `BtoCW` will always produce a word of $W_{n,t}$.

The function BtoCW is called with three integers n, t and δ as arguments and it returns a t-tuple $(\delta_1, \ldots, \delta_t)$ of non-negative integers. It is easy to check by induction, that we have $\delta_1 + \ldots + \delta_t \leq n + \delta - t$. Thus reading a binary stream B with BtoCW with the arguments $(n, t, 0)$ will return a t-tuple $(\delta_1, \ldots, \delta_t)$ whose sum is less or equal to $n - t$, which precisely corresponds to a word $W_{n,t}$. We will thus be able to encode any binary sequence of information into a sequence of elements of $W_{n,t}$. We might need to add a few bits (zeroes or random bits or whatever) in order to complete the last letter of $W_{n,t}$. In cryptography, the protocol should take great care of that to avoid leaking information on the last block, but that is not the point here.

E. Efficiency and complexity

At each call to the recursive procedure BtoCW, at least one bit of the input is read, so the complexity depends linearly of the input length. In practice and in average, there are $2t$ recursive calls. We ran the procedure BtoCW on randomly generated files, we measured the number of bits we had to read before producing a words of $W_{n,t}$. The average number of bits we can "put" in each element of $W_{n,t}$ is upper bounded by the entropy of $W_{n,t}$ equipped with a uniform distribution $H(W_{n,t}) = \log_2 \binom{n}{t}$. The ratio will measures the efficiency of the method.

We used this source coding algorithm in the implementation of Niederreiter's system (see §II-D). The measured information rate is given in Table II. The parameters proposed are typical of what is needed for cryptologic purposes [McE78], [Nie86], [CFS01].

		number of bits read					info. rate
n	t	min.	max.	aver.	$\log_2 \binom{n}{t}$	effi.	(Mbits/s)
1024	50	272	294	282.4	284.0	.9941	14.0
2048	29	208	223	215.0	215.9	.9957	17.3
4096	9	86	94	89.2	89.5	.9967	20.5
65536	9	122	129	125.2	125.5	.9976	18.3

TABLE II

SIMULATION RESULTS FOR VARIOUS VALUES OF (n, t)

We present in Table III a trade-off where we restrict values of d (given by best_d) to powers of two. This greatly simplifies the encoding process and gives a significant advantage in speed while the loss of efficiency is very limited.

Simulation were conducted in the same condition as in §II-D. With the parameters $(n, t) = (2048, 29)$, the running time for the 100 Mbytes file is 24 seconds instead of 194 in §II-D. The speed factor is 8. If we only look at the source coding part, the speed factor is more than 20 (8 seconds instead of 180).

		number of bits read					info. rate
n	t	min.	max.	aver.	$\log_2 \binom{n}{t}$	effi.	(Mbits/s)
1024	50	262	294	280.4	284.0	.9874	25.4
2048	29	202	224	213.7	215.9	.9897	33.0
4096	9	82	94	89.1	89.5	.9950	39.5
65536	9	119	131	125.1	125.5	.9964	22.0

TABLE III

SIMULATION RESULTS WHEN d IS LIMITED TO POWERS OF TWO FOR VARIOUS VALUES OF (n, t)

REFERENCES

[BB87] J. M. Borwein and P. B. Borwein. *Pi and the AGM*. John Wiley and sons, 1987.

[Bre76a] R. P. Brent. *The Complexity of Multiple-Precision Arithmetic, Complexity of Computational Problem Solving*. Univ. of Queensland Press, 1976.

[Bre76b] R. P. Brent. Fast multiple-precision evaluation of elementary functions. *Journal of the ACM*, 23:242–251, 1976.

[CFS01] N. Courtois, M. Finiasz, and N. Sendrier. How to achieve a McEliece-based digital signature scheme. In C. Boyd, editor, *Asiacrypt 2001*, number 2248 in LNCS, pages 157–174. Springer-Verlag, 2001.

[Cov73] T. Cover. Enumerative source encoding. *IEEE Transactions on Information Theory*, 19(1):73–77, January 1973.

[Gol66] S. Golomb. Run-length encoding. *IEEE Transactions on Information Theory*, 12(3):399–401, July 1966.

[McE78] R. J. McEliece. A public-key cryptosystem based on algebraic coding theory. *DSN Prog. Rep.,* Jet Prop. Lab., California Inst. Technol., Pasadena, CA, pages 114–116, January 1978.

[Nie86] H. Niederreiter. Knapsack-type crytosystems and algebraic coding theory. *Prob. Contr. Inform. Theory*, 15(2):157–166, 1986.

[Sen95] N. Sendrier. Efficient generation of binary words of given weight. In Colin Boyd, editor, *Cryptography and Coding ; proceedings of the 5th IMA conference*, number 1025 in LNCS, pages 184–187. Springer-Verlag, December 1995.

[Sen02a] N. Sendrier. *Cryptosystèmes à clé publique basés sur les codes correcteurs d'erreurs*. Mémoire d'habilitation à diriger des recherches, Université Paris 6, 2002.

[Sen02b] N. Sendrier. On the security of the McEliece public-key cryptosystem. In M. Blaum, P.G. Farrell, and H. van Tilborg, editors, *Information, Coding and Mathematics*, pages 141–163. Kluwer, 2002. Proceedings of Workshop honoring Prof. Bob McEliece on his 60th birthday.

[Sid94] V. M. Sidel'nikov. A public-key cryptosystem based on Reed-Muller codes. *Discrete Mathematics and Applications*, 4(3):191–207, 1994.