Fast Fibonacci Encoding Algorithm*

Jiří Walder, Michal Krátký, and Jan Platoš

Department of Computer Science
VŠB–Technical University of Ostrava
17. listopadu 15, Ostrava–Poruba, Czech Republic
{jiri.walder,michal.kratky,jan.platos}@vsb.cz

Abstract. Data compression has been widely applied in many data processing areas. Compression methods use variable-length codes with the shorter codes assigned to symbols or groups of symbols that appear in the data frequently. Fibonacci code, as a representative of these codes, is often utilized for the compression of small numbers. Time consumption of encoding as well as decoding algorithms is important for some applications in the data processing area. In this case, efficiency of these algorithms is extremely important. There are some works related to the fast decoding of variable-length codes. In this paper, we introduce the Fast Fibonacci encoding algorithm; our approach is up-to 4.6$\times$ more efficient than the conventional bit-oriented algorithm.

1 Introduction

Data compression has been widely applied in many data processing areas. Various compression algorithms have been developed for processing text documents, images, video, etc. In particular, data compression is of the foremost importance and has been well researched as it is presented in excellent surveys [13, 18].

Various codes have been applied for data compression [14]. In contrast with fixed-length codes, statistical methods use variable-length codes, with the shorter codes assigned to symbols or groups of symbols that have a higher probability of occurrence. People who design and implement variable-length codes have to deal with these two problems: (1) assigning codes that can be decoded unambiguously and (2) assigning codes with the minimum average size.

In some applications, a prefix code is required to code a set of integers whose length is not known in advance. The prefix code is a variable-length code that satisfies the prefix property. As we know, the binary representation of integers does not satisfy this condition. In other words, the size n of the set of integers has to be known in advance for the binary representation since it determines the code size as $1 + \lfloor \log_2 n \rfloor$. Fibonacci coding is distinguished as a suitable coding for a compression of small numbers [13].

* Work is partially supported by Grants of GACR No. P202/10/0573 and SGS, VŠB–Technical University of Ostrava, No. SP/2010138.

The time consumption of decompression is more critical than the time of compression; therefore, efficient decompression algorithms were studied in many works for the decompression of data structures [15, 6, 16] or text files [12, 3]. In the case of physical implementation of database systems, retrieval of compressed data structure's pages may be more efficient than retrieval of uncompressed pages due to the fact that the cost of decompression is lower than the cost of page accessing in the secondary storage [16, 2].

Since fast decoding algorithms have not yet been known, variable-length codes have not been used to compression of data structures, and, in generally, in the data processing area. The first effort of the fast decoding algorithm for Fibonacci codes of order ≥ 2 has been proposed in [7, 8]. We studied fast decoding algorithms of various variable-length codes in our previous work [17]. The fast encoding algorithms for the Fibonacci code yet not has been studied. In the case of data structures, pages are decompressed during every reading from the secondary storage into the main memory or items of a page are decompressed during every access to the page. If insert or update operations are considered, data compression becomes more significant.

In this article, we present Fast encoding algorithm of the Fibonacci of order 2 code. In Section 2, we describe the conventional Fibonacci of order 2 encoding algorithm. In Section 3, we provide a theoretical background of the fast encoding algorithm based on Fibonacci right shift and Encoding-Interval table. In Section 4, experimental results are presented and the proposed algorithm is compared to the conventional approach. In the last section, we conclude this paper and outline future works.

2 Fibonacci Coding

In this section, the theoretical background of Fibonacci of order 2 is briefly described. This universal code introduced by Apostolico and Fraenkel in [1] is based on the Fibonacci numbers [10]. The sequence of Fibonacci numbers is defined as follows:

$$F_i = F_{i-1} + F_{i-2} \text{ , for } i \geq 1,$$
$$\text{where } F_{-1} = F_0 = 1.$$

Definition 1. *(Fibonacci binary encoding and computation of its value)*

Let $F(n) = a_0 a_1 a_2 \ldots a_p$ be the Fibonacci binary encoding of a positive integer n. The value of the Fibonacci binary encoding, denoted $V(F(n))$, is defined as follows:

$$V(F(n)) = n = \sum_{i=0}^{p} a_i F_i \ (a_i \in \{0, 1\}, 0 \leq i \leq p)$$

In the Fibonacci binary encoding, each bit represents a Fibonacci number F_i. Such a number has the property of not containing any sequence of two consecutive 1-bits [1]. This property is utilized for the construction of the Fibonacci code $\mathcal{F}(n)$ of number n. Fibonacci code $\mathcal{F}(n)$ maps n onto a binary string so that the string ends with a sequence of two consecutive 1-bits. The Fibonacci codes for some integers are shown in Table 1.

Table 1. Examples of Fibonacci codes for some integers

n	$F(n)$							$\mathcal{F}(n)$
1	1							11
2	0	1						011
3	0	0	1					0011
4	1	0	1					1011
5	0	0	0	1				00011
6	1	0	0	1				10011
7	0	1	0	1				01011
8	0	0	0	0	1			000011
16	0	0	1	0	0	1		0010011
32	0	0	1	0	1	0	1	00101011
i	0	1	2	3	4	5	6	
F_i	1	2	3	5	8	13	21	

In Algorithm 1, we see how the conventional bit-oriented algorithm encodes a positive integer n into a Fibonacci code. This algorithm outputs the encoded number into Fn and its length into LFn. Due to the fact that bits of Fibonacci coding are encoded in the reverse order we must use the temporary Fn variable. Bits in the variable are right-shifted in each inner loop.

3 Fast Fibonacci Encoding Algorithm

The main issue of the conventional encoding algorithm is handling encoded numbers in the bit-by-bit manner. To design a fast encoding algorithm, encoded numbers are separated into segments larger than one bit. Similar principles have been utilized in fast decoding algorithms [17, 8, 9]. The separation of encoded numbers utilizes the Fibonacci right shift operation introduced in [17, 9]. Individual segments are then encoded by the precomputed *Encoding-Interval table.* When all individual segments are encoded, they are put together into the complete code. The Fibonacci right shift and Encoding-Interval table are presented in Section 3.1. The fast algorithm is explained in Section 3.2.

3.1 Fibonacci Shift Operation and Encoding-Interval Table

The Fibonacci shift operation introduced in [17, 9] is required for the bit manipulation in fast encoding and decoding algorithms. In this paper, we introduce an efficient computation of this operation.

```
input  : n, a positive integer
output: Fn, encoded number by Fibonacci code of order 2 with LFn length
p ← 0 ;
while $F_p \leq$ n do p ←p +1;
p ← p − 1;
Fn ← 1;
LFn ← 1;
while p ≥ 0 do
    Fn ← Fn << 1;
    if $F_p \leq$ n then
        Fn ← Fn | 1;
        n ← n − $F_p$;
    end
    LFn ←LFn +1;
    p ← p − 1;
end
```

Algorithm 1: Conventional encoding algorithm for the Fibonacci code of order 2

Definition 2. *Fibonacci shift operation*

Let $F(n) = a_0a_1a_2 \ldots a_p$ be a Fibonacci binary encoding, k be an integer, $k \geq 0$. The k-th Fibonacci left shift $F(n) <<_F k$ is defined as follows:

$$F(n) <<_F k = \overbrace{00\ldots0}^{k} a_0a_1a_2 \ldots a_p$$

Fibonacci right shift is defined as is follows:

$$F(n) >>_F k = a_ka_{k+1}a_{k+2} \ldots a_p$$

We utilize k-Fibonacci right shifts for the separation of numbers into segments. We do not need all k-Fibonacci right shifts, we only need multiplies of the segment size S. If we consider 32 bit-length integers than the length of the largest code is $L(F(2^{32})) = 46$; therefore, $k \in \{0, 8, 16, 24, 32, 40\}$ for $S = 8$ and $k \in \{0, 16, 32\}$ for $S = 16$.

The conventional approach to the calculation of Fibonacci right shift works in the following steps:

1. Compute $F(n)$ for the n value according to Definition 1.
2. Binary-shift the bits of $F(n)$ by k: $F(n) >> k$.
3. Compute the value of the shifted number $F(n) >> k$ according to Definition 1.

The Fibonacci right shift operation is time consuming since the Fibonacci value computation in Steps 1 and 3 requires a summarization of Fibonacci numbers for 1-bits; therefore, we utilize the Encoding-Interval table for the fast computation.

The Encoding-Interval table allows separating of numbers into segments with the k-th Fibonacci right shift and then direct translation of segments into Fibonacci codes. The size of the Encoding-Interval table depends on the size of the segment S. The segment size is usually one byte, i.e. $S = 8$. When we use larger segments the length of the Encoding-Interval table grows; on the other hand, the encoding becomes faster. There are $F_S - 1$ codes which can fit into one segment; it means $F_8 - 1 = 54$ codes which can fit into the 8 bit-length segment and $F_{16} - 1 = 2,583$ codes for the 16 bit-length segment.

Therefore, the total size of the Encoding-Interval table is:

$$table\_length = (F_S - 1) \times \left\lceil \frac{2^b}{S} \right\rceil$$

where b is the number of bits of the largest code.

Each line of the Encoding-Interval table is then built for all $F(n)$ codes which can fit into one segment and for all k-th Fibonacci right shifts. Each line includes the following values (see Table 3 for some examples):

- $F(n)$ – the Fibonacci code stored in the segment.
- $L(F(n))$ – the bit-length of the Fibonacci code
- k – the parameter k of the Fibonacci right shift operation
- n – the value stored in the actual segment $n = V(F(n))$
- $\langle n_{\min}, n_{\max} \rangle$ – an interval of numbers before the shift operation; this number is formally defined as follows: $\forall x \in \langle n_{\min}, n_{\max} \rangle : V(F(x) \;>>\; _F\, k) = \; n.$

This table can be used for the computation of the k-th Fibonacci right shift. For each x value we need to pass through the table to find the correct line where $x \in \langle n_{\min}, n_{\max} \rangle$. In this line we can directly read the shifted value $V(F(x)) >>_F \; k = n$ and also the corresponding Fibonacci code $F(n)$. Obviously, we need to pass only lines with correct k-th Fibonacci right shift.

To be able to compute a shifted value as fast as possible, we must consider the properties of the Fibonacci code. Let F_i denote the i-th term of the Fibonacci sequence then we can express each Fibonacci number by

$$F_i = \left\| b\phi^i \right\|$$

where ϕ is the well-known golden ratio [11] and a is the coefficient of the dominating term [9]. In the case of Fibonacci of order 2 the value $\phi = \frac{1+\sqrt{5}}{2} \approx 1.6180$ and $b = \frac{3\sqrt{5}+5}{10}$ [4]. The Fibonacci sequence calculated according to this formula is shown in Table 2.

Table 2. Examples of the Fibonacci sequence calculation

i	$a\phi^i$	$\lVert a\phi^i \rVert$
0	1.17	1
1	1.89	2
2	3.07	3
3	4.96	5
4	8.02	8
5	12.98	13
6	21.01	21
7	33.99	34
8	55	55
9	89	89
10	144	144

If we utilize this property, Fibonacci right shift $F(n) >>_F k$ is approximately calculated by the following equation:

$$a_0a_1a_2\dots a_p >>_F k = \sum_{i=0}^{p} a_iF_i >>_F k = \sum_{i=0}^{p} a_i \left\| b\phi^i \right\| >>_F k$$
$$\approx \sum_{i=0}^{p} a_ib\phi^i >>_F k = \frac{\sum_{i=0}^{p} a_ib\phi^i}{\phi^k} = \sum_{i=0}^{p} a_ib\phi^{i-k} \approx \sum_{i=0}^{p} a_i \left\| b\phi^{i-k} \right\| =$$
$$\sum_{i=0}^{p} a_iF_{i-k} = \sum_{i=-k}^{p-k} a_{i-k}F_i = \sum_{i=0}^{p-k} a_{i-k}F_i = a_ka_{k+1}a_{k+1}\dots a_p.$$

The shift operation result is not calculated precisely due to the fact that we ignore twice rounds. We can simply rewrite the approximate computation of the Fibonacci right shift as follows:

$$V(F(n) >>_F k) \approx \left\| \frac{V(F(n))}{\phi^k} \right\|$$

The maximum error of the estimation is 1; by experiments we found, when the estimation misses, it is overestimated; therefore, the correct value is less by 1. To check if the estimation is correct we compare the estimation with a range in the Encoding-Interval table. If the check fails, we simply subtract the 1 value and receive the correct right shift value. In other words, this estimation decreases the linear complexity of the table scan to at most two attempts.

Example 1. First, let us consider $V(F(n)) = n = 265$; $F(n) = 001010100001$. We calculate the estimation of the 8-th Fibonacci right shift as follows:

$$V(F(265) >>_F 8) \approx \left\| \frac{265}{\phi^8} \right\| = \left\| \frac{265}{1.6180^8} \right\| = \left\| \frac{265}{46.9787} \right\| = \|5.4493\| = 5$$

Table 3 shows some lines of the Encoding-Interval table for the 8-th Fibonacci right shift. After checking the 5-th line of the table we see that the value lies in the interval $< 233; 287 >$; therefore, $V(0001) = 5$ is the correct value of the 8-th Fibonacci right shift.

Second, let us consider $V(F(n)) = n = 280$; $F(n) = 000001010001$. We calculate the estimation:

$$V(F(280) >>_F k) \approx \left\| \frac{280}{46.9787} \right\| = \|5.9601\| = 6$$

After checking the interval in the 6-th line of the Encoding-Interval table we see that the value not lies in the interval $< 288; 321 >$; therefore, the estimation is not correct and the correct value is less by 1. The correct value of the 8-th Fibonacci right shift of 280 is $V(0001) = 5$.

Table 3. Examples of the Encoding-Interval table for the 8-th and 16-th Fibonacci right shifts

n	$F(n)$	$L(F(n))$	k	n_{min}	n_{max}	n	$F(n)$	$L(F(n))$	k	n_{min}	n_{max}
1	1	1	8	55	88	1	1	1	16	2,584	4,180
2	2	2	8	89	143	2	2	2	16	4,181	6,764
3	4	3	8	144	198	3	4	3	16	6,765	9,348
4	5	3	8	199	232	4	5	3	16	9,349	10,945
5	8	4	8	233	287	5	8	4	16	10,946	13,529
6	9	4	8	288	321	6	9	4	16	13,530	15,126
7	10	4	8	322	376	7	10	4	16	15,127	17,710
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
27	73	7	8	1,275	1,308	27	73	7	16	59,898	61,494
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
46	149	8	8	2,173	2,206	46	149	8	16	102,085	103,681
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

3.2 Fast Fibonacci Encoding Algorithm

The fast encoding algorithm is shown in Algorithm 2. This algorithm utilizes the previously proposed Encoding-Interval table denoted by EIT. Due to the fact that the bits of the Fibonacci code are stored in the reverse order we need to write the bits of segments in the reverse order as well. Encoded segments or their parts are stored in a specific position of the result Fn by the $SetValueOfSegment$ function.

In Algorithm 2, n is encoded by Fibonacci coding; the result is stored in the Fn, the length is stored in the LFn. Short numbers with the code lower than F_8 are directly encoded by the Encoding-Interval table (see Lines 3–5). For larger numbers we first need to find the maximal k of the Fibonacci right shift (see Line 7). After the number is separated with k-Fibonacci right shift, it is encoded by the Encoding-Interval table and stored into the highest segment with position $k/8$ (see Lines 8–10). In Line 11, we subtract the minimal range of the encoded segment from the n. In Lines 12–20, all other segments except the lower one are separated by Fibonacci right shifts and they are encoded by the Encoding-Interval table. The length of the encoded number is increased by the segment size in Line 19. The lowest segment is encoded in Line 21. Finally, in Lines 23–24, the delimiter is put at the end of the encoded number.

```
   input  : n, a positive integer
   output: Fn, number encoded by Fibonacci code of order 2 with the LFn
           length
 1 Fn ← 0 ;
 2 k ← 8 ;
 3 if n < $F_k$ then
 4     LFn ← EIT[k ][Number].LFn;
 5     SetValueOfSegment(Fn,0,EIT[k ][Number].Fn);
 6 else
 7     while n < $F_{k+8}$ do  k ← k +8;
 8     n ← n $>>_F$ k;
 9     LFn ← 8+ EIT[k ].LFn;
10     SetValueOfSegment(Fn,k/8,EIT[k ][n ].Fn);
11     n ← n −EIT[k ][n ].Nmin;
12     while k > 8 do
13         k ← k −8;
14         if n $\geq F_k$ then
15             n ← n $>>_F$ k;
16             n ← n − EIT[k ][n ].Nmin;
17             SetValueOfSegment(Fn,k/8,EIT[k ][n ].Fn);
18         end
19         LFn ←LFn +8;
20     end
21     SetValueOfSegment(Fn,0,EIT[k ][Number].Fn);
22 end
23 SetBit(Fn,LFn,1);
24 LFn ←LFn +1;
```

Algorithm 2: Fast encoding algorithm for the Fibonacci code of order 2

Example 2. Encoding of the number 17327 is depicted in Figure 1. The value of the Fibonacci code stored in all segments is $V(F(17327)) = 17327$. Encoding of the highest segment is depicted in Figure 2(a). After the 16-th Fibonacci right shift of the value $F(17327)$, i.e. $F(17327) >>_F 16$, we obtain the shifted Fibonacci code $F(7)$. The value after the shift is depicted in Line 3. It represents the Segment 2 value. We directly access the line 7 of the Encoding-Interval table by the 16-th Fibonacci right shift and we directly find the code $F(7) = 10$ with the length $L(F(7)) = 4$. The result is in the range $\langle 15127; 17710\rangle$. The lower bound of the range is depicted in Line 2. The value of Segments 0 and 1 is obtained by subtracting the lower bound of the range from the $V(F(17327))$ number, i.e Segments 0 and 1 stores $V(F(17327)) - 15127 = V(F(2200))$. This encoding is carried out in Lines 8–11 of Algorithm 2.

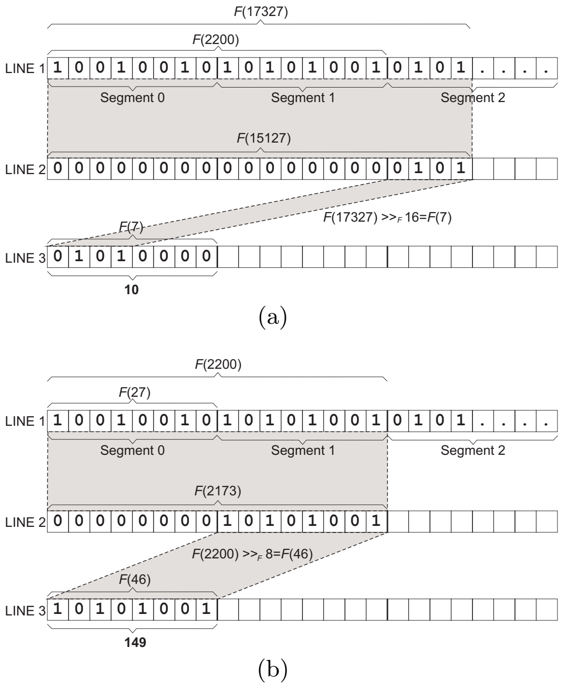

Fig. 1. Example of Fast Fibonacci encoding

Further encoding is depicted in Figure 2(b). The value of the Fibonacci code stored in Segments 0 and 1 is $V(F(2200)) = 2200$. After the 8-th Fibonacci right shift of the value $F(2200)$, i.e. $F(2200) >>_F 8$, we obtain the shifted Fibonacci code $F(46)$. The value after the shift is depicted in Line 3. It represents the separated value from Segment 1. We directly access the line 46 of the Encoding-Interval table by the 8-th Fibonacci right shift and we directly find the code $F(46) = 149$ with the length $L(F(46)) = 8$. The result is in the range $\langle 2173; 2206\rangle$. The lower bound of the range is depicted in Line 2. The value of Segment 0 is obtained by subtracting the lower bound of the range from $V(F(2200))$ number, i.e Segment 0 includes $V(F(2200)) - 2173 = V(F(27))$. This encoding is carried out in Lines 12–20 of Algorithm 2.

The last Segment 0 with the value $V(F(27))$ is directly encoded into $F(27) = 73$ with the length $L(F(27)) = 7$ in Line 21 of Algorithm 2. We access the line 27 of the Encoding-Interval table for any shift to obtain this value, because we do not need any following subtraction.

Finally, the 1-bit delimiter is added in position $8+8+L(F(7))+1 = 8+8+4 = 12$ of the encoded number (see Lines 23–24 of Algorithm 2).

4 Experimental Results

The proposed Fast Fibonacci encoding algorithm has been tested and compared with the conventional algorithm. The algorithms' performance has been tested for various test collections. The tests were performed on a PC with dual core Intel 2.4GHz, 3 GB RAM using Windows 7 32-bit.

The test collections used in experiments have the same size: $10,000,000$ numbers. The proposed algorithm is universal and it may be applied for arbitrary numbers > 0. However, we worked with numbers $\leq 4,294,967,295$, it means the maximal value is the value of the 32 bit-length binary number. Tested collections are as follows:

- 8-bit – a collection of random numbers ranging from 1 to 255
- 16-bit – a collection of random numbers ranging from 256 to 65,535
- 24-bit – a collection of random numbers ranging from 65,536 to 16,777,215
- 32-bit – a collection of random numbers ranging from 16,777,216 to 4,294,967,295
- ALL - a collection of random numbers ranging from 1 to 4,294,967,295

Table 4. Fast Fibonacci encoding times and speedup ratios for different random collections for conventional and fast algorithms with different segment sizes

Algorithm / Collection	**Conventional**	**Fast $S = 8$**		**Fast $S = 16$**	
	Time [ms]	**Time** [ms]	**Speedup** [times]	**Time** [ms]	**Speedup** [times]
8-bit	1,327	553	2.4	248	5.4
16-bit	2,399	889	2.7	547	4.4
24-bit	3,538	1,375	2.6	865	4.1
32-bit	4,539	1,829	2.5	992	4.6
ALL	4,547	1,808	2.5	992	4.6
Avg.	**3,270**	**1,291**	**2.5**	**729**	**4.6**

We performed tests for segment sizes $S = 8$ and $S = 16$. We ran each test 10 times and calculated average values. Results of all tests are depicted in Table 4 and Figure 2. The Fast Fibonacci encoding algorithm is approximately $2.6\times$ faster than the conventional approach for the segment size $S = 8$ and $4.6\times$

faster for the segment size $S = 16$. The encoding times linearly increase with the bit-length of encoded numbers but the speedup ratio is not influenced by this increasing.

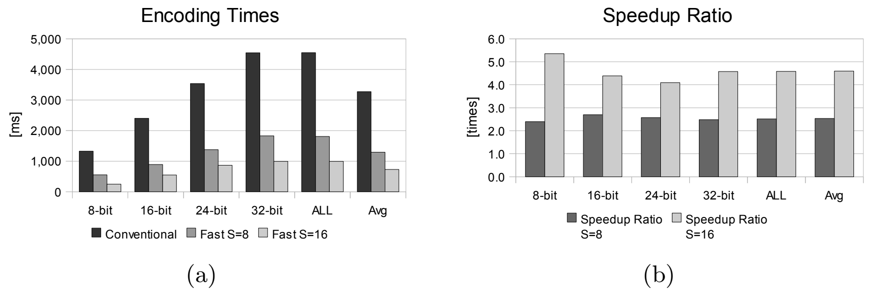

Fig. 2. (a) Encoding times for random collections and conventional and fast algorithms. (b) Speedup ratios between conventional and fast algorithms for random collections.

5 Conclusion

In this paper, the fast encoding algorithm for the Fibonacci code of order 2 is introduced. We introduced the effective implementation of Fibonacci right shift which is utilized by the encoding algorithm for separating integers into segments. The segments are directly translated into the Fibonacci code by the Encoding-Interval table. The improvement depends on the segment size used for the separation of the encoded numbers. For the segment size $S = 8$ (it means one byte), the Fast Fibonacci encoding is up-to $2.6\times$ more efficient than the conventional algorithm. For the larger segment of two bytes in size (it means $S = 16$), the proposed algorithm is up-to $4.6\times$ more efficient than the conventional algorithm. In our future work, we want to develop fast encoding algorithms for other universal codes like Elias-delta [5], Fibonacci code of order 3 [1], and so on.

References

1. A. Apostolico and A. Fraenkel. Robust Transmission of Unbounded Strings Using Fibonacci Representations. *IEEE Transactions on Information Theory*, 33(2):238–245, 1987.
2. R. Bača, J. Walder, M. Pawlas, and M. Krátký. Benchmarking the Compression of XML Node Streams. In *Proceedings of the BenchmarX 2010 International Workshop, DASFAA, Accepted.* Springer-Verlag, 2010.
3. T. C. Bell and I. H. Witten. *Text Compression.* Prentice Hall, 1990.

4. R. A. Dunlap. *The Golden Ratio and Fibonacci Numbers.* World Scientific Publishing Co. Pte. Ltd., 1997.
5. P. Elias. Universal Codeword Sets and Representations of the Integers. *IEEE Transactions on Information Theory*, IT-21(2):194–203, 1975.
6. J. Goldstein, R. Ramakrishnan, and U. Shaft. Compressing Relations and Indexes. In *Proceedings of the 14th International Conference on Data Engineering, ICDE 1998*, page 370, Los Alamitos, CA, USA, 1998. IEEE Computer Society.
7. S. T. Klein. Fast Decoding of Fibonacci Encoded Texts. In *Proceedings of the International Data Compression Conference, DCC'07*, page 388, Washington, DC, USA, 2007. IEEE Computer Society.
8. S. T. Klein and M. K. Ben-Nissan. Using Fibonacci Compression Codes as Alternatives to Dense Codes. In *Proceedings of the International Data Compression Conference, DCC'08*, pages 472–481, Washington, DC, USA, 2008. IEEE Computer Society.
9. S. T. Klein and M. K. Ben-Nissan. On the Usefulness of Fibonacci Compression Codes. *Accepted in Computer Journal, 2009*, 2009.
10. Leonardo of Pisa (known as Fibonacci). *Liber Abaci.* 1202.
11. M. Livio. *The Golden Ratio: The Story of Phi, the World's Most Astonishing Number.* Broadway, January 2003.
12. H. Plantinga. An Asymmetric, Semi-adaptive Text Compression Algorithm. In *Proceedings of the International Data Compression Conference, DCC 1994*. IEEE Computer Society, 1994.
13. D. Salomon. *Data Compression The Complete Reference.* Third Edition, Springer–Verlag, New York, 2004.
14. D. Salomon. *Variable-length Codes for Data Compression.* Springer-Verlag, 2007.
15. H. Samet. Data Structures for Quadtree Approximation and Compression. *Communications of the ACM archive*, 28(9):973–993, September 1985.
16. J. Walder, M. Krátký, and R. Bača. Benchmarking Coding Algorithms for the R-tree Compression. In *Proceedings of the Dateso 2009 Annual International Workshop on Databases, Texts, Specifications and Objects*, pages 32–43. CEUR Workshop Proceedings, Volume: 471, 2009.
17. J. Walder, M. Krátký, R. Bača, J. Platoš, and V. Snášel. Fast Decoding Algorithms for Variable-Lengths Codes. *Submitted in Information Science*, January, 2010.
18. I. H. Witten, A. Moffat, and T. C. Bell. *Managing Gigabytes, Compressing and Indexing Documents and Images, 2nd edition.* Morgan Kaufmann, 1999.