Integer Codes Correcting Spotty Byte Asymmetric Errors

Aleksandar Radonjic and Vladimir Vujicic

Abstract—**In short range optical networks channel errors occur due to energy losses. Upon transmission, they mostly manifest themselves as spotty byte asymmetric errors. In this paper we present a class of codes that can correct these errors. The presented codes use integer and lookup table operations, which makes them suitable for software implementation. In addition, if needed, the proposed codes can be interleaved without delay and without using any additional hardware.**

Index Terms—**Integer codes, error correction, asymmetric errors, look-up table.**

I. Introduction

In wireless networks it is impossible to predict the polarity of channel errors. Also, due to the presence of EMI/RFI, it is hard to draw any general conclusion regarding the error behavior: in some case random errors dominate, while in other cases burst errors are the rule rather than the exception. In optical networks without optical amplifiers (e.g. local and access networks) the situation is quite different. First, in these networks the number of received photons never exceeds the number of transmitted ones. Hence, upon transmission only $1 \rightarrow 0$ transitions can occur [1]. Second, due to immunity to EMI/RFI, channel errors affect only small number of bits. This phenomenon was first reported in [2], and then confirmed by many other experiments [3], [4]. In all these studies it was shown that 96.9% to 99.9% of all errors are t-bit errors ($t \leq 3$) confined to one or two adjacent bytes.

Despite these facts, the codes correcting asymmetric errors ($1 \rightarrow 0$ errors) are mainly designed for use in non-networked environments (see [5], [6] and references therein). A notable exception are Bose-Al-Bassam codes [7] that can correct all asymmetric errors confined to a b-bit byte. This capability is achieved by using two checksums: one, which is obtained by XORing all data bytes, and other, which incorporates the information about the number of 1's within each data byte. Although this solution resembles the Fletcher's algorithm [8], it includes operations at the bit level. Hence, it cannot be efficiently implemented on general purpose hardware. On the other hand, in local and access networks this type of implementation is highly preferred, since the network nodes (PCs, servers, routers, switches, OLU/OTN units, etc.) already possess high computing power [9], [10].

The authors are with the Institute of Technical Sciences of the Serbian Academy of Sciences and Arts, 11000 Belgrade, Serbia (e-mail: sasa_radonjic@yahoo.com; vujicicv@yahoo.com).

Motivated by this, in this paper we present a new class of integer codes. The most attractive feature of these codes is the ability to encode/decode using $3k$ operations per codeword, where k denotes the number of data bytes. Besides this, the proposed codes use simple method to correct spotty byte asymmetric errors. In essence, it is about search-and-compare strategy, where the non-zero syndrome is compared with table entries. As will be seen, this solution is similar to routing [9], and thus highly suitable for software implementation.

II. Codes Construction

A. Encoding and Decoding Procedures

Let $Z_{2^b-1} = \{0, 1,\ldots, 2^b - 2\}$ be the ring of integers modulo $2^b - 1$, and let us suppose that the data are divided into k b-bit bytes (Fig. 1). In addition, let C_i and C_{k+1} be integers such that $C_i \in Z_{2^b-1} \setminus \{0,1\}$ and $C_{k+1} = -1$.

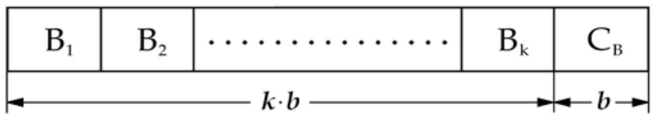

Fig. 1. The codeword structure of the proposed codes.

As in [11], the check-byte C_B is computed using

$$C_B = [C_1 \cdot B_1 + \cdots + C_k \cdot B_k] \ (\mathrm{mod}\ 2^b - 1) = \sum_{i=1}^{k} C_i \cdot B_i \ (\mathrm{mod}\ 2^b - 1) \tag{1}$$

At the receiver, the decoder will perform the same calculation

$$C_{\hat{B}} = [C_1 \cdot \hat{B}_1 + \cdots + C_k \cdot \hat{B}_k] \ (\mathrm{mod}\ 2^b - 1) = \sum_{i=1}^{k} C_i \cdot \hat{B}_i \ (\mathrm{mod}\ 2^b - 1) \tag{2}$$

after which the syndrome S will be formed

$$S = [C_{\hat{B}} - \hat{C}_B] \ (\mathrm{mod}\ 2^b - 1) \tag{3}$$

Obviously, when $S \neq 0$, the codeword is corrupted by noise.

B. Necessary and Sufficient Conditions

Definition 1. *An error is called a spotty byte asymmetric error or t/b asymmetric error if t or fewer bits within a b-bit byte are in a $1 \rightarrow 0$ error, where $1 \leq t < b$.*

Definition 2. *Let $e_m = \{2^{x_1} + \cdots + 2^{x_m}\}$ be a positive integer, where $0 \leq x_1 < \cdots < x_m < b$ and $1 \leq m \leq t$. Then, the set of syndromes corresponding to t/b asymmetric errors is defined as*

$$\varepsilon = \left\{ \bigcup_{m=1}^{t} \bigcup_{i=1}^{k+1} (-C_i \cdot e_m) \ (\mathrm{mod}\ 2^b - 1) : \ 1 \leq t < b \right\} \tag{4}$$

With these definitions we can prove the following theorem.

Theorem 1. *The codes defined by (1)-(4) can correct all t/b asymmetric errors if and only if there exist k mutually different coefficients* $C_i \in Z_{2^b-1} \setminus \{0,1\}$ *such that*

$$\left|\varepsilon\right| = (k+1)\cdot\sum_{m=1}^{t}\binom{b}{m}$$

where $\left|\varepsilon\right|$ denotes the cardinality of ε.

Proof. From (4) it is clear that the set ε can be expressed as

$$\varepsilon = \bigcup_{i=1}^{k+1} X_i$$

where

$$X_1 = \left\{\bigcup_{m=1}^{t}\left(-C_1\cdot e_m\right)\left(\text{mod } 2^b-1\right) : 1 \le t < b\right\}$$

$$\vdots$$

$$X_k = \left\{\bigcup_{m=1}^{t}\left(-C_k\cdot e_m\right)\left(\text{mod } 2^b-1\right) : 1 \le t < b\right\}$$

$$X_{k+1} = \left\{\bigcup_{m=1}^{t}\left(e_m\right)\left(\text{mod } 2^b-1\right) : 1 \le t < b\right\}$$

Obviously, if the coefficients C_i are chosen in such a way that each one multiplied (modulo 2^b - 1) by each e_m yields a different result it is clear that

$$X_1 \cap \cdots \cap X_k \cap X_{k+1} = \varnothing$$

$$\left|X_1\right| = \cdots = \left|X_k\right| = \left|X_{k+1}\right|.$$

In that case, the set ε will have

$$\left|\varepsilon\right| = \left|X_1\right| + \cdots + \left|X_k\right| + \left|X_{k+1}\right| = (k+1)\cdot\left|X_{k+1}\right| = (k+1)\cdot\sum_{m=1}^{t}\binom{b}{m}$$

nonzero elements. □

In order to illustrate the applicability of Theorem 1, we show results of a computer-search for some codes with parameters $t \le 3$ (Tables 1 and 2).

TABLE I
NUMBER OF COEFFICIENTS FOR CODES WITH PARAMETERS $T \le 3$ AND $B \le 16$.

	$b = 4$	$b = 6$	$b = 8$	$b = 10$	$b = 12$	$b = 14$	$b = 16$
$t = 1$	2	8	29	98	334	1160	4079
$t = 2$	0	1	1	5	8	17	29
$t = 3$	0	0	1	1	1	5	14

TABLE II
COEFFICIENTS FOR CODES WITH PARAMETERS $T = 3$, $B \le 32$ AND $K \le 64$.

$b = 16$							
2	15	71	89	143	179	377	593
1379	1499	2441	2477	2877	3467		
$b = 24$							
2	15	31	71	83	89	139	141
157	167	173	189	203	269	277	281
303	305	331	339	429	475	543	573
583	895	921	1065	1115			
$b = 32$							
2	15	31	71	83	89	101	119
127	139	141	143	149	157	163	167
173	177	179	181	189	191	199	203
211	223	227	229	233	239	251	253
263	269	271	277	281	283	305	307
313	317	331	339	349	353	359	361
367	373	379	383	389	395	397	401
409	421	431	433	443	463	465	467

C. *Error Control Procedure*

As mentioned, the key idea behind the proposed codes is to exploit high computing power of the network nodes. Such an approach drastically reduces implementation costs, since it does not require a dedicated hardware (the encoding/decoding circuits) as in the case of classical codes. Given this, let us suppose that the encoder and decoder are multicore processors (MPs) [9], [10]. In addition, let us assume that the decoder uses a lookup table (LUT$_2$) size of $\left|\varepsilon\right| \times \left[2\cdot b + \left\lceil log_2(k+1)\right\rceil\right]$ bits. Unlike the LUT$_1$, which stores the coefficients C_i, this table is generated using (4). Its aim is to describe the relationship between the nonzero syndrome (element of the set ε), error location (i) and error vector (e) (Fig. 2).

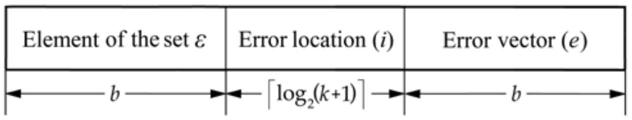

Fig. 2. Bit-width of one LUT$_2$ entry.

Bearing this in mind, it is easy to see that the main task of the decoder is to find the entry where the first b bits match that of the syndrome S. For that purpose, the decoder must perform a series of table lookups (similar to routing [9]). In the end, after n_{TL} table lookups, the decoder will declare failure ($S \notin \varepsilon$) or execute one of the following operations ($S \in \varepsilon$):

- *for t/b asymmetric errors within the i-th data byte*

$$B_i = [\hat{B}_i + e]\,(\text{mod } 2^b-1),\ 1 \le i \le k;$$
$$e = [e_m]\,(\text{mod } 2^b-1),\ 1 \le m \le t \le b-1; \tag{5}$$

- *for t/b asymmetric errors within the check-byte*

$$C_{\text{B}} = [\hat{C}_{\text{B}} + e]\,(\text{mod } 2^b-1);$$
$$e = [e_m]\,(\text{mod } 2^b-1),\ 1 \le m \le t \le b-1; \tag{6}$$

Although the procedure described above is very simple, it is clear that its efficiency depends on the number of table lookups. For this reason it is very important that the elements of ε are sorted in increasing order. In that case it is possible to use binary search algorithm, which requires n_{TL} table lookups ($1 \le n_{\text{TL}} \le \left\lfloor log_2\left|\varepsilon\right|\right\rfloor + 2$) [12].

Example 1. Let b = 8, m = 2, k = 1 and C_1 = 2. According to Theorem 1, the LUT$_2$ will have $\left|\varepsilon\right|$ = 72 entries. Given this, suppose that we want to transmit 8 bits of data, D = 10010011. In that case, after calculating the value of the check-byte C_{B}

$$C_{\text{B}} = [C_1\cdot B_1]\,(\text{mod } 2^b-1) = [2\cdot 147]\,(\text{mod } 255) = 39$$

the codeword C_{W} = 10010011 00100111 will have 16 bits. Now, let us analyze the following scenarios.

Case 1: Suppose that during data transmission an error on the 4th and 7th bit has occurred ($\hat{C}_{\text{W}}$ = 100$\underline{0}$00$\underline{0}$1 00100111). In that case, the decoder will calculate

$$C_{\hat{\text{B}}} = [C_1\cdot\hat{B}_1]\,(\text{mod } 2^b-1) = [2\cdot 129]\,(\text{mod } 255) = 3$$

$$S = [C_{\hat{\text{B}}} - \hat{C}_{\text{B}}]\,(\text{mod } 2^b-1) = [3-39]\,(\text{mod } 255) = 219$$

and check whether the value S = 219 belongs to the set ε (Table 3). After completing this task, it will perform error correction by using:

$$B_1 = [\hat{B}_1 + e]\,(\text{mod } 2^b-1) = [129+18]\,(\text{mod } 255) = 147.$$

TABLE III
LOOK-UP TABLE (LUT$_2$) FOR INTEGER (16, 8) DECODER.

	Element of the set ε	*i*	*e*		*Element of the set ε*	*i*	*e*		*Element of the set ε*	*i*	*e*		*Element of the set ε*	*i*	*e*
1	1	2	1	**19**	36	2	36	**37**	128	2	128	**55**	221	1	17
2	2	2	2	**20**	40	2	40	**38**	129	2	129	**56**	222	1	144
3	3	2	3	**21**	48	2	48	**39**	130	2	130	**57**	223	1	16
4	4	2	4	**22**	63	1	96	**40**	132	2	132	**58**	231	1	12
5	5	2	5	**23**	64	2	64	**41**	136	2	136	**59**	235	1	10
6	6	2	6	**24**	65	2	65	**42**	144	2	144	**60**	237	1	9
7	8	2	8	**25**	66	2	66	**43**	159	1	48	**61**	238	1	136
8	9	2	9	**26**	68	2	68	**44**	160	2	160	**62**	239	1	8
9	10	2	10	**27**	72	2	72	**45**	175	1	40	**63**	243	1	6
10	12	2	12	**28**	80	2	80	**46**	183	1	36	**64**	245	1	5
11	16	2	16	**29**	95	1	80	**47**	187	1	34	**65**	246	1	132
12	17	2	17	**30**	96	2	96	**48**	189	1	33	**66**	247	1	4
13	18	2	18	**31**	111	1	72	**49**	190	1	160	**67**	249	1	3
14	20	2	20	**32**	119	1	68	**50**	191	1	32	**68**	250	1	130
15	24	2	24	**33**	123	1	66	**51**	192	2	192	**69**	251	1	2
16	32	2	32	**34**	125	1	65	**52**	207	1	24	**70**	252	1	129
17	33	2	33	**35**	126	1	192	**53**	215	1	20	**71**	253	1	1
18	34	2	34	**36**	127	1	64	**54**	219	1	18	**72**	254	1	128

Case 2: Let us assume that during data transmission an error on the 11th bit has occurred ($\hat{C}_W$ = 10010011 $00\underline{0}00111$). Similar to the previous case, after calculating

$$C_{\hat{B}} = [C_1 \cdot \hat{B}_1] \pmod{2^b - 1} = [2 \cdot 147] \pmod{255} = 39$$

$$S = [C_{\hat{B}} - \hat{C}_B] \pmod{2^b - 1} = [39 - 7] \pmod{255} = 32$$

the decoder will conclude that the value S = 32 indicates an error within the check-byte (Table 3). On the basis of this information, it will perform the error correcting by using

$$C_B = [\hat{C}_B + e] \pmod{2^b - 1} = [154 + 32] \pmod{255} = 186.$$

III. INTERLEAVING AND IMPLEMENTATION STRATEGY

A. Interleaving

One of the most important features of the proposed codes is that they can be interleaved without delay. In order to illustrate how this can be done, suppose that we need to transmit $2 \cdot k$ b-bit bytes. In that case, instead of one, we will have two check-bytes. These check-bytes will be calculated as follows:

$$C_{B1} = \sum_{i=1}^{k} C_i \cdot B_{2i-1} \pmod{2^b - 1} \tag{7}$$

$$C_{B2} = \sum_{i=1}^{k} C_i \cdot B_{2i} \pmod{2^b - 1} \tag{8}$$

At the receiver, the decoder will perform identical operations

$$C_{B1} = \sum_{i=1}^{k} C_i \cdot B_{2i-1} \pmod{2^b - 1} \tag{9}$$

$$C_{B2} = \sum_{i=1}^{k} C_i \cdot B_{2i} \pmod{2^b - 1} \tag{10}$$

after which two syndromes will be formed

$$S_1 = [C_{\hat{B}1} - \hat{C}_{B1}] \pmod{2^b - 1} \tag{11}$$

$$S_2 = [C_{\hat{B}2} - \hat{C}_{B2}] \pmod{2^b - 1} \tag{12}$$

If we compare (1)-(3) and (7)-(12) we can conclude that the same encoder/decoder can be used for both non-interleaved and interleaved codes. Namely, in the case of non-interleaved codes there is only one check-byte. For its generation, the encoder/decoder sequentially uses the coefficients C_1, C_2, ... , C_k. In the case of interleaved codes we have two check-bytes. From (7)-(10) we see that the values of these check-bytes are always updated alternately. This, obviously, corresponds to the situation in which the encoder/decoder sequentially uses the coefficients C_1, C_1, C_2, C_2, ..., C_k, C_k. Hence, there is no delay during the encoding/decoding process.

B. Implementation Strategy

From the above it is clear that the en/decoder uses integer and LUT operations. Since these operations are supported by all processors, it is interesting to discuss how the proposed codes can be implemented on 8-core processors (EPs) (Fig. 3).

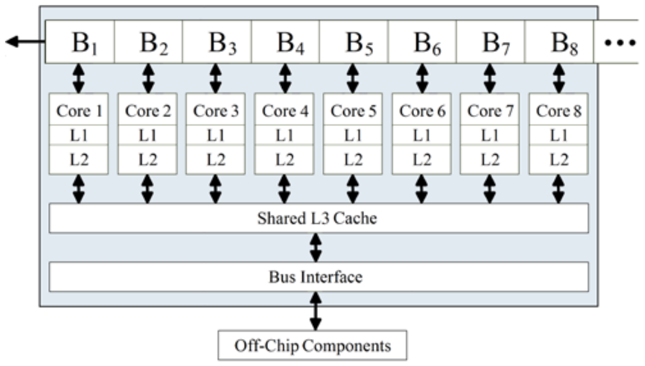

Fig. 3. Block diagram of 8-core processor.

First of all, recall that each processing core has at least one integer unit and two private caches: L1 and L2. The L1 has a small size (up to 64 KB) and very low access latency (1-5 clock cycles) [10]. The L2, on the other hand, is somewhat slower (8-15 clock cycles), but much larger (up to 512 KB) [10]. Finally, all cores have access to the shared L3 cache. This cache has the highest access latency (25-50 clock cycles), and largest storage capacity (up to 32 MB) [10].

Now, let us go back to the encoding, decoding and error control procedures. They can be briefly described as follows:

- *Encoding procedure*
  Step 1. Read the C_i's from a LUT$_1$ and compute the C_B according to (1).
- *Decoding procedure*
  Step 1. Read the C_i's from a LUT$_1$ and compute the $C_{\hat{B}}$ according to (2).
  Step 2. Calculate the syndrome S with (3).
- *Error control procedure*
  Step 1. If $S = 0$, the codeword is error-free.
  Step 2. If $S \neq 0$, perform a binary search of a LUT$_2$ to find the error vector (e) and error location (i).
  Step 3. If $S \in \varepsilon$, execute (5) or (6). Else, declare a failure.

To implement these procedures, both LUTs must be placed in appropriate caches. In the case of a LUT$_1$, the memory requirements are less than 1 KB (Table 4). Hence, this table should be placed into the L1 cache. As far as LUT$_2$ is concerned, we need to allocate up to 3.17 MB of memory (Table 4). This means that we do not have any other option, but to store this table in the L2/L3 cache.

TABLE IV
LOOK-UP TABLE SIZES FOR CODES WITH PARAMETERS $T = 3$ AND $B = 32$.

Code	Encoder	Decoder		
	LUT$_1$	LUT$_1$	LUT$_2$	
	Size	Size	Size	# of Table Lookups
(1024, 992)	124 B	124 B	1.51 MB	$1 \le n_{TL} \le 19$
(1056, 1024)	128 B	128 B	1.58 MB	$1 \le n_{TL} \le 19$
(2048, 2016)	252 B	252 B	3.07 MB	$1 \le n_{TL} \le 20$
(2080, 2048)	256 B	256 B	3.17 MB	$1 \le n_{TL} \le 20$

IV. COMPARISON WITH THE BEST S_bAEC CODES

From [2], [3], [4] it is easy to conclude that codes with the parameter $t = 3$ provide nearly the same level of reliability as the single b-bit byte asymmetric error correcting (S_bAEC) codes. For that reason we will compare our codes with the best S_bAEC codes [7]. First, let us focus on the parameters of both codes. According to [7], the S_bAEC codes use $b + \log_2(k)$ check bits. This means that the S_{16}AEC codes require up to 23 check-bits to protect data words of length $K \le 2048$ bits. On the other hand, we know that the proposed codes always have b check-bits. In addition, from Table 2 it is clear that data words of length $K \ge 1024$ bits can be protected only by using the codes with 32 check-bits.

From a practical point of view, however, this limitation is insignificant. Namely, during the software implementation the larger value of b is more preferable, since modern EPs operate in 64-bit mode. If we look at this from the perspective of the proposed codes, the benefit is clear: one processing core must perform $3k$ operations (Table 5) (k table lookups, k integer multiplications, k - 1 integer additions and one modulo reduction) per codeword. On the other hand, from [7] we know that each S_bAEC code consists of two check-bytes: the parity byte (PB) and the arithmetic residue check (ARC) byte. These bytes are calculated as

$$\mathrm{PB} = [\dot{B}_1 \oplus \cdots \oplus \dot{B}_k] \quad (13)$$

$$\mathrm{ARC} = [1 \cdot \dot{N}_1 + \cdots + k \cdot \dot{N}_k] \ (\mathrm{mod}\ p) \quad (14)$$

where $\oplus$ indicates the EXOR operation, $\dot{N}_i$ the number of 1's in transmitted/received byte $\dot{B}_i$, while p represents the smallest prime such that $(p - 1)/2 \ge k$. Since (13) and (14) are mutually independent, it is clear that "odd" processing cores must perform $k\cdot(b + 1)$ operations ($k\cdot(b - 1)$ binary additions, k integer multiplications, k - 1 integer additions and one modulo reduction) per codeword. As far as error control procedure is concerned, the S_bAEC codes also use LUTs. However, in this case a LUT is used to store multiplicative inverses of $\dot{N}_i$ with respect to modulo p.

TABLE V
COMPARISON OF PROPOSED CODES AND BOSE-AL-BASSAM CODES.

Main Characteristics	Proposed Codes	Bose-Al-Bassam Codes
Number of check-bits	b	$b + \log_2(k)$
Maximum number of data bytes	Not specified (depends on the results of a computer search)	The largest prime less than 2^k
Error correction capabilities	Correction up to three asymmetric errors within a b-bit byte	Correction of all asymmetric errors within a b-bit byte
Maximum number of en/decoding operations per core per codeword	$3\cdot k$	$(b + 1)\cdot k$
Number of error control operations	≤ 23	≤ 9
Size of error control table	≤ 3.17 MB	≤ 64 B

V. CONCLUSION

In this paper we presented a class integer codes capable of correcting spotty byte asymmetric errors. We have shown that the encoding/decoding procedures for these codes are very simple, whereas the error control algorithm resembles the routing process. Thanks to these features, the presented codes have a potential to be practically applied, especially in optical networks covering small areas.

REFERENCES

[1] P. Oprisan and B. Bose, "ARQ in Optical Networks," *Proc. IEEE Int'l Symp. Pacific Rim Dependable Computing*, pp. 251-257, Dec. 2001.
[2] CCITT Study Group XVIII Contribution D21, "Observations of Error Characteristics of Fiber Optic Transmission Systems," Jan. 1989.
[3] D. Mello, E. Offer and J. Reichert, "Error Arrival Statistics for FEC Design in Four-Wave Mixing Limited Systems," *Proc. Optical Fiber Communications Conference* (*OFC '03*), pp. 529-530, Mar. 2003.
[4] L. James, "Error Behaviour in Optical Networks", PhD thesis, Dept. of Engineering, University of Cambridge, 2005.
[5] T. Kløve, "Error Correcting Codes for the Asymmetric Channel", University of Bergen, Tech. Rep. 1809-0781, 1995.
[6] H. Kaneko and E. Fujiwara, "A Class of M-ary Asymmetric Symbol Error Correcting Codes for Data Entry Devices," *IEEE Trans. Computers*, vol. 53, no. 2, pp. 159-167, Feb. 2004.
[7] B. Bose and S. Al-Bassam, "Byte Unidirectional Error Correcting and Detecting Codes," *IEEE Trans. Computers*, vol. 41, no. 12, pp. 1601-1606, Dec. 1992.
[8] J. Fletcher, "An Arithmetic Checksum for Serial Transmission," *IEEE Trans. Communications*, vol. 30, no. 1, pp. 247-252, Jan. 1982.
[9] R. Giladi, *Network Processors: Architecture, Programming, and Implementation*, Elsevier, Inc., 2008.
[10] L. Johnsson, "Introduction to HPC Architecture," University of Houston, Jan. 2014.
[11] A. Radonjic and V. Vujicic, "Integer Codes Correcting Burst Errors within a Byte," *IEEE Trans. Computers*, vol. 62, no. 2, pp. 411-415, Feb. 2013.
[12] K. Mehlhorn and P. Sanders**,** *Algorithms and Data Structures: The Basic Toolbox*, Springer, 2008.