Binary Codes for Non-uniform Sources

(DCC-2005)

Alistair Moffat
and
Vo Ngoc Anh

Presented by:-
Palak Mehta 11-20-2006

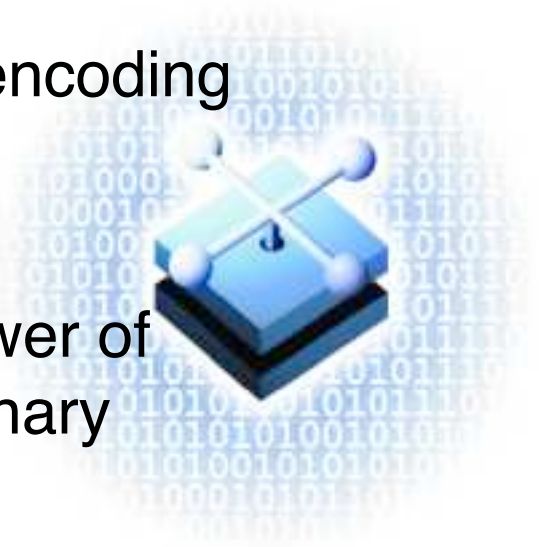
Basic Concept:

- In many applications of compression, decoding speed is at least as important as compression effectiveness.
- So here they present two coding methods that make use of fixed binary representations.
- They have all of the consequent benefits in terms of decoding performance.

Brief About:

- Unary Codes – It is an entropy encoding method that represents a natural number *n* with *n* ones followed by zero. Eg. 5 is represented as 111110.

- Elias _ Codes – They are universal codes encoding positive integers. To code:



  - Separate the integer into the highest power of 2 it contains (2^N) and the remaining N binary digits of the integer
  - Encode N in Unary; that is as N zeros followed by a one
  - Append the remaining N binary digits to this representation of N. Eg. $2^7 = 2^2+3 = 001\textit{11}$

Brief About (contd...):

- Golomb Codes – It is a form of entropy encoding that is optimum alphabets following geometric distributions, ie; when small values are vastly more common than large values. The code depends on choice of a parameter b.

- The first step is to compute the 3 quantities,
  q = int [n/b] ; r = [n%b] ; c = [$\log_2 b$],
  where n is the no. being encoded.

- Eg. Let b = 10 and so c = 4

r	0	1	2	q	0	1	2
code	000	001	010	code	0	10	110

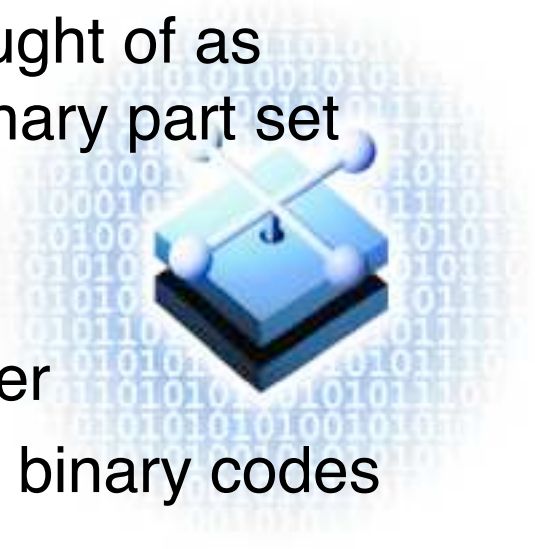
Thus, the Golomb Code for 22 will be 110*010*

The Vision:

- They introduced two coding methods that have following properties:
  - Based on simple binary codes and thus fast to decode
  - Sensitive to localized variations in probability distribution and thus capable of outperforming Huffman Codes
  - Parameterless, and thus still useful even when the message to be represented is small compared to the number of permitted alphabet symbols

Recursive Multi-span Selector:

- In a Binary Code, all codewords are of the same length
- Golomb and Elias Codes can also be thought of as binary codes, but with the length of the binary part set on a codeword by codeword basis.



- So the key idea here is that there are other possibilities between the two extremes of binary codes and
  elias codes
- Eg. Consider the following list of integers:
  13, 8,3,1,0,0,0,0,5,3,7,1,6

Recursive Multi-span Selector (Contd...):

- Transforming each value x to the corresponding bit length $[\log_2(x+1)]$ gives a sequence of selector values: 4,4,2,1,0,0,0,0,3,2,3,1,3

- As, contrast to Elias _ code, suppose that the list of selectors is grouped in blocks containing s_1 code, where s_1 is fixed in advance.

- So our example by assuming $s_1 = 2$ would be: 4,2,0,0,3,3,3 by taking maximum over pairs

Recursive Multi-span Selector (contd...):

- The matching codeword string would then be:
  1101 1000 11 01 . . . . 101 011 111 001 110

- We can handle the sequence of selectors recursively using the same exact method.

- Using s_2=2 as an example parameter at the next level, we get: 3,0,2,2 and with the bitstring as:
  100 010 . . 11 11 11

- Now further recursive passes result in the need for a single number to be transmitted as the first item in the message

Recursive Multi-span Selector (contd...):

- The number of items in the message is also required before decoding can be commenced, in order that the recursive structure can be recovered

- If $s_1 = s_2 = \ldots = s$ for some fixed value s, then the number of recursive levels required to code a message of m integers grows as $\log_s m$

- However, using $s_i = s$ is not necessarily ideal because the repeated taking of log on the values means that they quickly fall into compact and consistent range.

- Hence, instead of using $s_i=s$, we take $s_i=s^i$

Recursive Multi-span Selector (contd...):

- With this arrangement, the no. of recursions is $(\sqrt{\log_s m})/2$ and for s = 4 and m = 1,000,000 there are just 4 levels

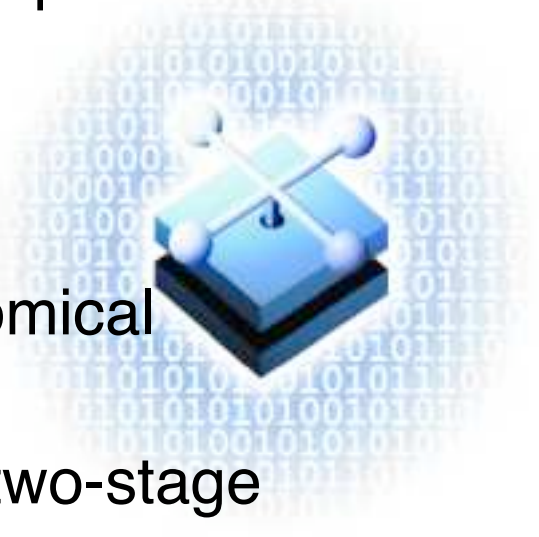
- This new method is called **RBUC(*s*)** (Recursive Bottom Up, Complete)

- Given the small number of plausible values for *s*, it is reasonable to extend the encoder and have it search over the possibilities and indicate to the decoder which values of *s* it will employ for this particular message

- This method is called **RBUC-B** (Recursive Bottom Up Complete, Best)

Top Down Evaluation:

- Suppose that it is known that the largest of the *m* message symbols can be coded in *b* bits
- Then clearly, the entire message can be represented in *m.b* bits
- So if the other *m*-1 values are also large, coding the message might be quite economical
- This observation suggests an alternative two-stage process



Top Down Evaluation (Contd...):

- In the first stage, a bottom-up hierarchy of maximum symbol values and coding costs is constructed in a tree structure with s-way branching at each node

  – During this stage, the *s* children of each node report to their parent the maximum number of bits required by any symbol plus the number of bits required to resolve that parameter for each of the *s* subtrees

Top Down Evaluation (Contd...):

- In the second top-down phase, information is percolated back down the tree from the root, and the decision is made at each node whether it is sensible to continue the recursive decomposition into *s* smaller subsequences or whether
  it is better to terminate

- An s-bit control variable is added at each decision point

- Each bit in the control variable corresponds to one of the *s* subtrees, and stipulates the relationship between codeword length

Top Down Evaluation (Contd...):

- Then the *k*th bit in the control variable is set to 0, if the *k*th subtree contains one or more symbols that require *b* bits, and is set to 1 if every symbol in that subtree can be stored in strictly less than *b* bits

- As a single exception, if all *s* bits in the control variable are on then this node is regarded as a "stopping" node and all of the integers controlled by the node are coded as *b* bit binary values

- So, at each node, decision is made, whether it is more economical to continue the recursive decomposition or to simply use a flat binary tree

Top Down Evaluation (Contd...):

- Here, it is explored, that an exact value of *b* is maintained at
  every node in the tree rather than an upper bound

- To do this, every one-bit in the control variable is augmented by a follow-up binary code of $[\log_2 b]$ bits, storing a number between 0 and b -1 inclusive, indicating the exact parameter for the corresponding subtree

- This method is called RTDP(s) Recursive Top Down, Partial), where s>1 is an integer indicating the uniform branching of the tree

Experiments:

File name	Origin	Total symbols	Maximum value	Self-information (bits/sym)
`Data1`	Character level BWT and MTF on the standard file `bible.txt`	4,047,397	123	1.99
`Data2`	Word level BWT and MTF on the standard file `bible.txt`	970,881	13,753	6.56
`Data3`	Character level BWT and MTF on 20 MB of newspaper text	20,971,525	198	2.13
`Data2`	Word level BWT and MTF on 20 MB of newspaper text	4,731,236	72,029	7.59
`Data5`	The *d*-gaps in the inverted index for 267 MB of newspaper text	41,389,467	173,252	6.76

Table 1: Test data used, and its origins.

EXPERIMENTS(CONTD...):

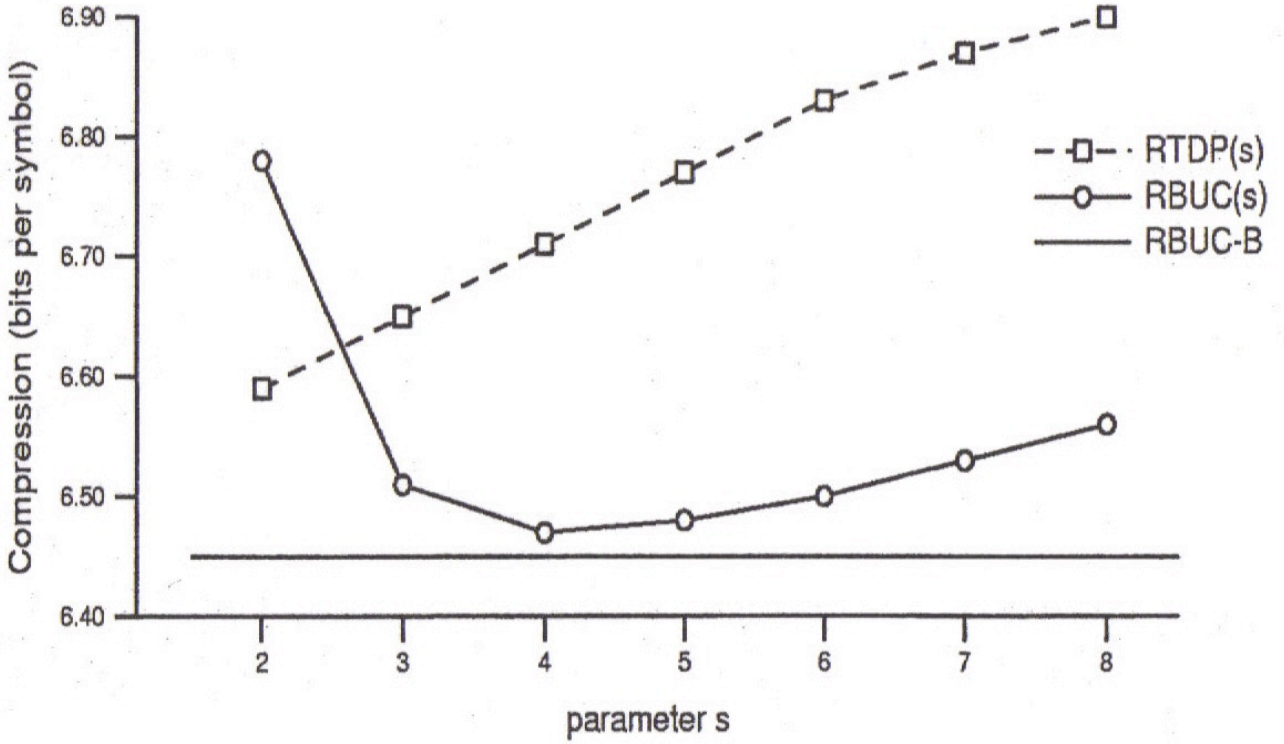


Figure 1: Compression effectiveness (bits per symbol) of coding schemes on Data5.

EXPERIMENTS(CONTD...):

Method	Data1	Data2	Data3	Data4	Data5
Golomb	2.07	9.21	2.20	11.26	8.53
interpolative	**1.70**	6.90	**1.77**	7.81	**6.03**
bytealigned	8.00	9.36	8.00	10.36	9.35
shuff	2.01	**6.68**	2.11	**7.68**	6.64
carryover-12	2.49	9.21	2.76	10.15	7.01
RBUC(4)	1.91	7.92	1.97	8.85	6.47
RBUC-B	1.88	7.51	1.93	8.29	6.45
RTDP(2)	2.02	7.18	2.06	7.85	6.61

Table 2: Compression effectiveness (bits per symbol) for five different data files.

EXPERIMENTS(CONTD...):

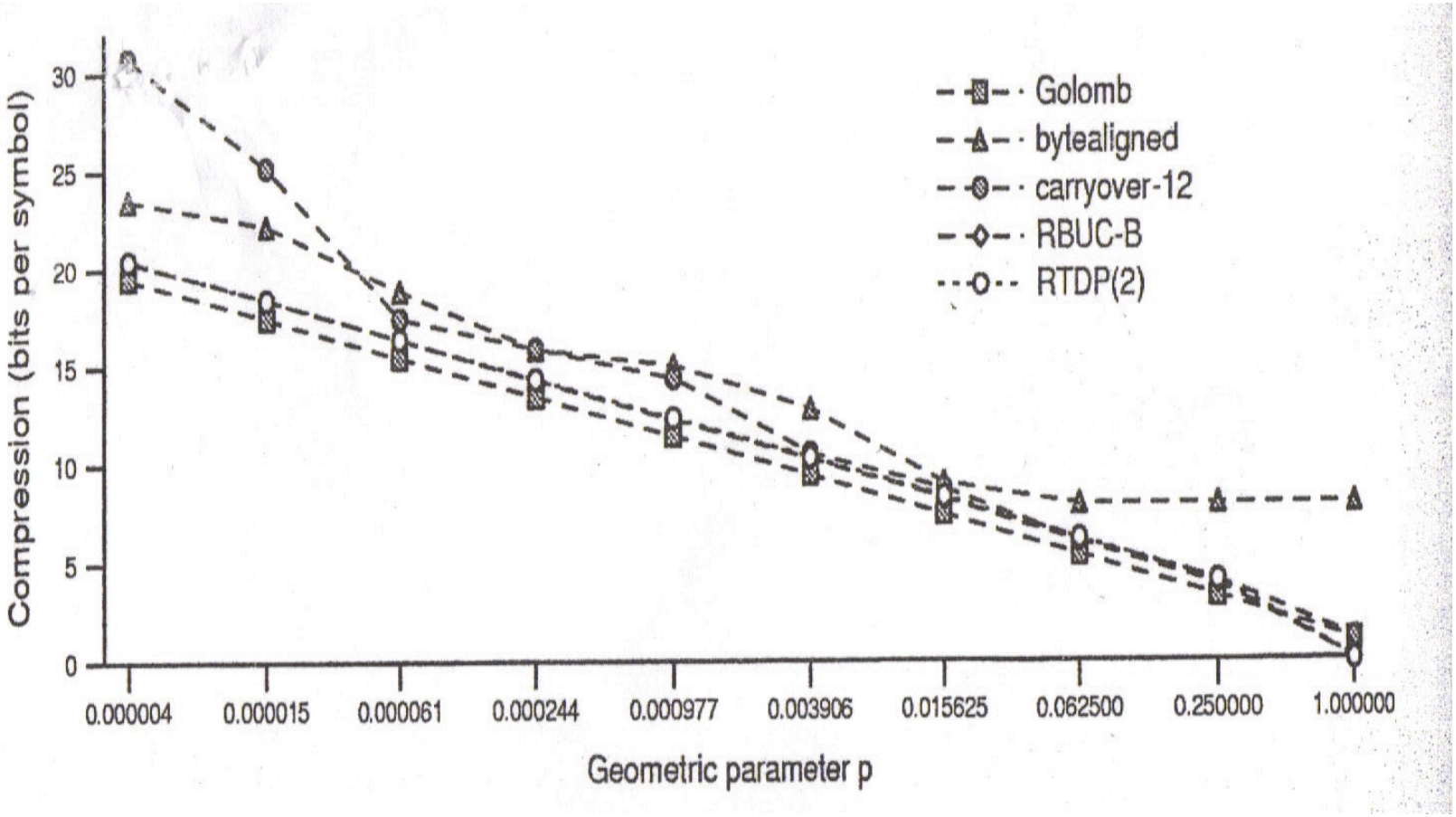


Figure 2 :Compression obtained by different coding schemes on synthetic geometric data parameterized by a probability p. Each point represents a message of 1,000,000 integers

EXPERIMENTS(CONTD...):

Method	Data1	Data2	Data3	Data4	Data5
bytealigned	0.062	0.022	0.305	0.112	0.694
shuff	0.153	0.051	0.781	0.258	1.734
carryover-12	**0.053**	**0.017**	**0.269**	**0.077**	**0.591**
RBUC(4)	0.073	0.023	0.375	0.103	0.822
RBUC-B	0.075	0.024	0.383	0.130	0.820
RTDP(2)	0.097	0.042	0.483	0.183	1.042

Table 3 : Decoding speed (measured as total elapsed time in seconds to decode the compressed data file and write an output), carried out using a dual 2.8 Ghz Intel Xeon 2 GB of RAM, twelve 146 GB SCSI disks in RAID-5 configuration and running Debian GNU/Linux

Performance in a retrieval system :

- The primary interest in the new methods is a representation for the inverted lists required by text retrieval systems.

- The fast performance of Google, for e.g., is in part a consequence of the fact that the inverted lists are stored compressed, and yet can still be processed quickly.

- To measure the usefulness of the new methods in realistic setting, they built a retrieval system for a collection of 18GB of text containing 1.2 million documents taken from the .gov domain as part of TREC project run by NIST

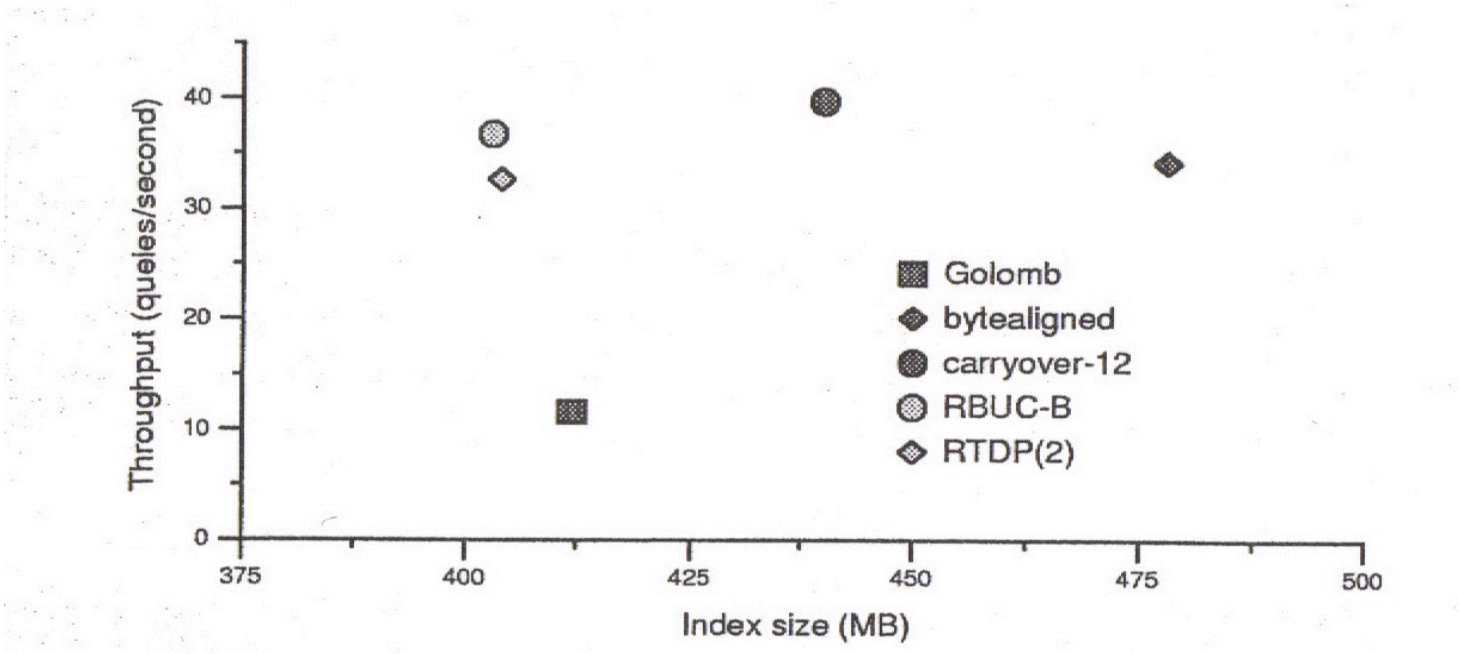
Performance in a retrieval system (contd...):

- Several different compression methods were used to represent the *d*-gaps. The actual index structure used in this retrieval system differs slightly. This variation makes the average *d*-gap larger and favors the byte-aligned code.

- The retrieval system implements a ranking heuristic that scores each document in the collection and returns the identifiers of the top 1000 documents according to that score.

- To test querying speed, a set of 10000 random queries were used, each drawn from a document in the collection

Performance in a retrieval system (contd...):



Fiqure 3 : Tradeoffs in effectiveness, measured as index size for an 18 GB document collection, and querying throughput, measured as queries per second on a dual 2.8 Ghz Intel Xeon 2 GB of RAM, twelve 146 GB SCSI disks in RAID-5 configuration and running Debian GNU/Linux. The 10,000 queries were generated randomly from the collection, and averaged three terms each

Conclusion:

- Two new binary based unparameterized codes are introduced that are applicable for decreasing probability distribution
- This codes give excellent compression and because they are sensitive to localized probability changes, are capable of outperforming entropy coders
- The also provide fast compression
- Software implementing RBUC-B method is available at:
  - www.cs.mu.oz.au/~alistair/rbuc/.

REFERENCES:

- Binary Codes for Non-Uniform Sources
- www.wikipedia.com