EXPERIMENTAL STUDY OF INDEXING DATA-STRUCTURES IN EXTERNAL MEMORY

TAN WEI KUN[1] AND DR KEN SUNG[2]

School of Computing, National University of Singapore
3 Science Drive 2, Singapore 117543

ABSTRACT

Indexing is often used to speed up searching of large amounts of data, however since this data is usually quite large, they are not able to fit into internal memory. However in external memory, operations on data are much slower than if the data was stored in internal memory. As such, it is of interest to us to reduce the amount of data that is needed to be retrieved from external memory. This leads us to the use of compression to allow the same amount of data to be represented in less space. Here we investigate the viability of a simple compressed index data structure scheme experimentally and mathematically.

1 INTRODUCTION

Due to the nature of large data collections, it is often the case that they cannot fit into internal memory and must be stored in external memory such as hard disks. This causes all operations on the data to be much slower due to the inherently slow nature of external memory. Thus, it is of interest to do compression on such data collections, either to reduce the collection to a size that can be stored within internal memory or to reduce the amount of raw data that has to be read from external memory while still maintaining a quick access/search time through indexing.

2 COMPRESSED INDEXING STRUCTURES

2.1 Ferragina & Venturini's Storage Scheme

In the compression scheme proposed by Ferragina & Venturini (2007), the basic idea is to create a 2 level look-up table into the compressed file in order to allow constant time access to the part of the file as required. The compression scheme's lookup tables is shown in figure 2.

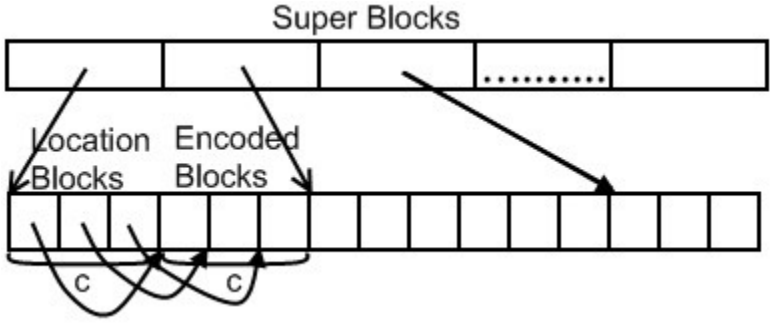

Figure 1. Data Representation of Ferragina & Venturini's Storage Scheme

[1] Student

[2] Supervisor

The encoding scheme employed here is to build a frequency table of the words that appear in the file. Each word is a sequence of bits of a certain length to be determined. Using this frequency table, the scheme will encode the most frequently appearing words as the bit sequence 0, 1, 00, 01, 10 and so on. This will have to be stored as a frequency table in the file as well.

Further, we will need to store the length of the encoded bit and this is done on the 2nd level size blocks table, as represented by the size block in Diagram 1. A group of such blocks will be grouped together and a 1st level super block will point to the first block of the group's 2nd level size block tables.

2.2 Sadakane & Grossi's Scheme

In Sadakane & Grossi (2006), they have proposed a way to further compress succinct data structures while still maintaining the same time complexity for the set of primitive operations supported by the original data structures, this means that any operation that requires t time in the original data structure will require at most $O(t)$ time in the new compressed data structure proposed in this paper.

The basic idea is to store some data that would allow us to access the individual words or symbols of the LZ78 Compression Scheme in O(1) time. Given the original data structure S, we can generate a compressed string Z using the LZ78 Scheme which is optimal in space. By storing two additional sets of arrays, a method has been found which allows for constant time decoding of any words in the compressed string Z.

3 Implementation

3.1 Implementation Details

In our implementation, we group each O(log n) location/data blocks into a single cluster that is referenced by a superblock. A large constant value in this case would increase the size of the location blocks while reducing the number of superblocks, while a small constant value will achieve the opposite effect. We will try to determine experimentally a suitable constant value for this.

3.2 Modified Implementation

In our modification, we have replaced the Probabilistic Table Encoding by Hoffman codings. This ensures optimal symbol-by-symbol encoding that performs much better than the original algorithm in terms of space complexity at the expense of increasing the time complexity to access a single symbol. In effect, we have replaced the location and data blocks in the original structure and replaced them with data blocks containing Hoffman codes.

4 Experimental Results

4.1 Testing Methodology

In order to test this data structure, we use sample files from 3 distinctly different file types. These 3 file types are text data files, bitmap image files, wav sound files. These file types are in their raw and uncompressed form.

4.2 Comparison

	Cluster Size	Hoffman	Probabilistic	Ratio
Bitmap File	1	13,019kb	18,072kb	0.7204
Text File	1	15,745kb	22,064kb	0.7136
Wav File	1	3,414kb	4,879kb	0.6998

Table 1. Comparison of Hoffman vs Probabilistic

If we compare the space taken up by both structures in Table 4, we can see that the Hoffman Code version outperforms the original version significantly in terms of space occupied all the time, ranging from 21-37% better. This shows the shortcomings of the original probabilistic algorithm.

4.3 Compression Ratio

	Original File	Probabilistic	Hoffman	.zip	.rar	Remarks
Bitmap File	12,657kb	16,103kb	11,050kb	3,760kb	2,253kb	Windows Default Wallpaper (2400x1600)
Text File	16,882kb	19,710kb	13,391kb	3,732kb	27kb	Repetitive Text of Size 5kb
Wav File	3,405kb	4,302kb	2,835kb	1,592kb	1,300kb	Sound Clip of Random Noises

Table 2. Comparison of Compressed File Sizes using different compression techniques(Excluding the superblocks for our data structures)

By observing the vast difference in the compression ratio between our entropy encoding techniques and the commercially used variable-length encoding, it is obvious that entropy encoding is very limited in its ability to compress any data significantly. As such, it would seem that Sadakane & Grossi's Scheme that makes use of the LZ78 Compression which is a variable-length encoding technique would perform much better.

5 Asymptotic Analysis

5.1 Ferragina & Venturini's Storage Scheme

To store this compression scheme, the total space required would be

$$O(|\sum|^b \log n + n/(b\log n) \log |V| + n/b \log\log^2 n + |V|) \quad (1)$$

bits. Where $\sum$ is the set of alphabets, $b=\text{floor}(1/2 \log_{|\sum|} n)$ is the word/block size, n is the length of the string, V is the encoded string, S is the original string and C is the final compressed structure. The access time to a single word is O(e) and access time to *l* consecutive words is O(*l*e) where e is the longest encoding string for a single symbol.

To further break down the space complexity, we require $O(|\sum|^b \log n)$ bits to store the lookup table, $O(n/(b\log n) \log |V|)$ to store the superblocks, $O(n/b \log\log^2 n)$ to store the location blocks and $O(|V|)$ bits to store the compressed string.

We attempt to determine mathematically the compression ratio required to yield a compressed index that is smaller than the original file. In order to simplify the calculation, we fix the alphabet size $\sum = \{0,1\}$ and we ignore the superblock structure which is actually superfluous if we are only interested in the compress and uncompress operations. This gives us the space complexity of

$$O(2^b \log n + n/b \log\log^2 n + |V|) = O(|C|) \quad (2)$$

bits to represent the data structure.

First, we allow r to represent the compression ratio, thus $|V| = rn$, since we are interested in the threshold at which the compressed index C is smaller than the original string, we have to satisfy

$$|C| < n. \quad (3)$$

We have

$$2^b \log n + n/b \log\log^2 n + rn < n, \quad (4)$$

by substituting $b=\text{floor}(1/2 \log_{|\sum|} n)$, we have

$$\sqrt{n} + 2n\log\log^2 n/\log n + rn < n \quad (5)$$

which can be simplified to

$$1 - 1/\sqrt{n} - 2\log\log^2 n/\log n > r. \quad (6)$$

File Size	**1 Kilobyte (2^{13})**	**1 Megabyte (2^{23})**	**1 Gigabyte (2^{33})**	**1 Terabyte (2^{43})**	**1 Petabyte (2^{53})**
Required Ratio	-0.1497	0.2123	0.3886	0.4952	0.5677

Table 3. Minimum Compression Ratio required for different file sizes. Negative denotes impossible to achieve smaller compression value.

Compression Ratio	**0.5**	**0.6**	**0.7**	**0.8**	**0.9**
Approximate File Size Required	2^{44} bits	2^{59} bits	2^{86} bits	2^{144} bits	2^{336} bits

Table 4. Approximate File Size required for smaller compressed file given the compression ratio.

From our experimental results, we can see that the compression ratio ranges from 0.59 to 0.68 for the 3 files that we tested. Based on the mathematical calculations, it turns out that in order for the compressed file in this case to be smaller than the original file with a compression ratio of 0.5, we would require a file size of approximately 2^{44} bits and if we have a compression ratio of 0.6, we require a file size of approximately 2^{59} bits.

This shows that this scheme is largely impractical due to the low compression ratio that is required for any space savings to be achieved from using this structure.

5.2 Modified Ferragina & Venturini's Storage Scheme

By modifying the original scheme, we achieve a space complexity of

$$O(|\sum|^b \log n + n/bc \log |V| + |V|) \quad (7)$$

bits. However the access time to access a single word in the data structure is increased to O(e logn), where e is the longest Hoffman code used to represent a word. To access *l* consecutive words, the access time would be O(*l*(e + logn)).

As before, we ignore the bits required to store the superblocks and concentrate only on the lookup tables and the encoded bits. This gives us a space complexity of

$$|\sum|^b \log n + |V| \quad (8)$$

bits to store our structure, we allow |V| = rn where r is the compression ratio. As before, we would like to have

$$|C| < n; \quad (9)$$

We have

$$2^b \log n + rn < n, \quad (10)$$

by substituting b=floor(1/2 $\log_{|\sum|} n$), we have

$$\sqrt{n} + rn < n \quad (11)$$

which can be simplified to

$$1 - 1/\sqrt{n} > r. \quad (12)$$

File Size	**2^4 bits**	**2^8 bits**	**2^{12} bits**	**2^{16} bits**	**2^{20} bits**
Required Ratio	0.7500	0.9375	0.9844	0.9961	0.9990

Table 5. Minimum Compression Ratio required for different file sizes.

As we can see, the required compression ratio is actually quite high in contrast to the previous scheme. If we consider that the experimental results yielded compression ratios ranging from 0.793 to 0.873, we can see that this scheme is quite practical

5.3 Sadakane & Grossi's Scheme

The space complexity of this scheme is

$$O(n(\log|\sum| + \log\log_{|\sum|} n + k)/\log_{|\sum|} n + |V|), \quad (13)$$

where $\sum$ is the set of alphabets, n is the length of the string and V is the encoded string.

When $|\sum| = 2$ and $k = O(\log\log n)$, the data structure will have size

$$O(n\log\log n/\log n + |V|) \quad (14)$$

bits.

6 Conclusion

It is readily apparent that the data structure proposed by Ferragina & Venturini (2006) is not practical in the real world due to the large file size required before the compression ratio is reasonably achievable to compress the file smaller than its original size, we require a file size

of 2^{144} bits to compress a file with at most 0.8004 compression ratio. However if we replace the encoding method with Hoffman Codings, the required file size before the compression ratio becomes practical is quite small, approximately 2^7 bits would only require a compression ratio of 0.9116.

7 REFERENCE

[1] Richard Cole, Lee-Ad Gottlieb and Moshe Lewenstein (2004), "Dictionary Matching and Indexing with Errors and Don't Cares.", *In Proc. Of Symposium on Theory of Computing*, 91-100

[2] C. Meek, J. M. Patel and S. Kasetty (2003), "OASIS: An online and accurate technique for local-alignment searches on biological sequences", *In VLDB*, 910-921

[3] David A. Huffman (1957),"A Method for the Construction of Minimum-Redundancy Codes", *Proceedings of the I.R.E* 40, 1098-1101

[4] J. Ziv and A. Lempel (1978), "Compression of individual sequences via variable-rate coding". *IEEE Trans. Inform. Theory*, IT-24(5):530–536

[5] Kunihiko Sadakane and Roberto Grossi (2006), "Squeezing succinct data structures into entropy bounds". *In Procs ACM-SIAM SODA*, 1230-1239

[6] M. Burrows and D. J. Wheeler (1994), "A Block-sorting Lossless Data Compression Algorithm", *Technical Report 124, Digital Equipment Corporation*

[7] Paolo Ferragina and Giovanni Manzini (2001), "An experimental study of an opportunistic index", *ACM-SIAM symposium on Discrete algorithms*, 269-278.

[8] Paolo Ferragina and Rossano Venturini (2007), "A simple storage scheme for strings achieving entropy bounds", *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, 690-696.

[9] S. Kurtz (1999), "Reducing the space requirement of suffix trees", *Software – Practice and Experience*, 29(13): 1149-1171

[10] T. F. Smith and M. S. Waterman (1981), "Identification of common molecular subsequences", *Journal of Molecular Biology*, 147:195-197

[11] T.W. Lam, W.K. Sung, S.L. Tam, C.K. Wong, and S.M. Yiu (2005), "Compressed Indexing and Local Alignment of DNA", *Bioinformatics*, 1-7.

[12] Wing-Kai Hon, Tak-Wah Lam, Rahul Shah, Siu-Lung Tam, and Jeffrey Scott Vitter (2007), "A Cache-Oblivious Index for Approximate String Matching", *Proceedings of the 16th Annual Conference on Combinatorial Pattern Matching (CPM '07)* , 40-51.