

# Boosting Pattern Matching Performance via $k$ -bit Filtering

M. Oğuzhan Külekci , Jeffrey Scott Vitter, and Bojian Xu

Texas A&M University  
Department of Computer Science & Engineering  
College Station, TX, USA

{kulekci,jsv,bojianxu}@cse.tamu.edu

**Abstract.** This study explores an alternative way of storing text files in a different format that will speed up the searching process. The input file is decomposed into two parts as filter and payload. Filter part is composed of most informative  $k$ -bits of each byte from the original file. Remaining bits form the payload. Selection of the most informative bits are achieved according to their entropy. When an input pattern is to be searched on the new file structure, same decomposition is performed on the pattern. The filter part of the pattern is queried in the filter part of the file following by a verification process of the payload for the matching positions. Experiments conducted on natural language texts, plain ascii DNA sequences, and random byte sequences showed that the search performance with the proposed scheme is on the average two times faster than the tested exact pattern matching algorithms.

## 1 Introduction

A file is a sequence of bytes stored in digital media, such as hard disks, DVDs, or flash memories. Currently text is stored on digital media as it would be written to an ordinary paper notebook. This study explores an alternative way of storing text files in a different format that will result in a boost in search performance, with the restriction that the size of the converted file will not exceed the original one.

Exact string matching, which is simply finding all occurrences of a given pattern on a text, is one of the deeply studied problems in computer science. The topic may be investigated in two classes as *online* and *offline* pattern matching [1]. Online methods create an index over the target data beforehand, while the offline methods [2] preprocess the input pattern for fast scanning over the text. The building blocks of indexing are subword graphs, suffix trees, and suffix arrays [3]. The cost of the gain via indexing is the large space consumption [4] of those structures in addition to the heavy procedure of their constructions in practice.

This study attempts to enhance the offline pattern matching performance via storing files in a different structure, given name *k-bit filter* file format. The conversion of an ordinary file into  $k$ -bit filter format is a light operation when

compared with the constructions of index structures and the resultant file is exactly of the *same size* as the original one.

In offline pattern matching, filtering is a powerful tool. While scanning the text, filtering algorithms such as *agrep* [5] and *q-hash* [6] compute the hash of the investigated window and compare against that of the pattern. Since hash collision does not necessarily require an exact match, a complete verification is to be done on those text positions reporting a match in the filtering phase.

The bits of bytes in a file have different entropies. For example, in an English text, the most significant bit of all characters is always zero as the printable characters are the first 128 characters in the ASCII table. This indicates that no information is carried out on that bit according to the information theory [7]. Thus, any time spent on comparing these bits is not much of use. The main idea of this study is to move the most informative bits of the bytes to the beginning of the file and then use those bits as a filter. Since that filter part will compromise a great portion of the file's information content, it acts as an effective filter and reduces the number of verification calls.

The work presented in this paper is bit-oriented rather than the classical byte-oriented approaches. Hence, bit vector matching is crucial throughout the study. A variant of the Külekci's BLIM [8] algorithm is used within this study to fulfill the requirements of the proposed filtering scheme in a fast and flexible manner. Some other recent bitvector matching algorithms can be found in [9–12].

## 2 *k*-bit Filtered File Format

Let file  $F$  of size  $n$  be a sequence of bytes denoted by  $F = s_1 s_2 s_3 \dots s_n$ , where each byte  $s_i$ ,  $1 \leq i \leq n$ , is composed of eight bits shown by  $s_i = b_1^i b_2^i \dots b_8^i$ .

The number of bits that will be extracted from each byte is denoted by  $k$ . Let set  $R$  contain the indices of the most informative  $k$  bits. The bits corresponding to the indices given in  $R$  are extracted from each byte  $s_i$  and are stored as a bit vector preserving the order of the bytes. This bit stream is placed at the beginning of the new file, and the remaining bits of each  $s_i$  are concatenated to this stream. Figure 1 denotes a sample  $k$ -bit filter file structure, assuming  $k = 2$ , and the indices of the most informative  $k$  bits are  $R = \{3, 5\}$ .

Filter Part	Payload Part
$b_3^1 b_5^1   b_3^2 b_5^2   \dots   b_3^n b_5^n$	$b_1^1 b_2^1 b_4^1 b_6^1 b_7^1 b_8^1   b_1^2 b_2^2 b_4^2 b_6^2 b_7^2 b_8^2   \dots   b_1^n b_2^n b_4^n b_6^n b_7^n b_8^n$
$k.n$ bits	$(8 - k).n$ bits

**Fig. 1.** Sample file  $F$  converted to  $k$ -bit filtered file structure, assuming  $k = 2$  and  $R = \{3, 5\}$ .

The first step while converting a file into  $k$ -bit filter structure is to find out the indices of the most informative  $k$  bits among the bytes of that file. It is a known

fact from the information theory [7] that the information carried by a sequence is inversely proportional to its compression ratio. With that in mind, let us assume eight sequences, each of which is composed of the bits appearing at positions 1 to 8 of each byte. Formally, let  $B_x = b_x^1 b_x^2 b_x^3 \dots b_x^8$ , for  $1 \leq x \leq 8$ . When these eight sequences are individually compressed and are sorted according to their sizes in descending order, this order also represents the amount of information content of the corresponding bits. For example, if the descending order of the sizes of the eight compressed  $B_x$  sequences is obtained as  $\{5, 3, 1, 6, 8, 7, 2, 4\}$ , then the most informative bit in each byte is the fifth, and the next most informative one is the third. If  $k = 2$  is given, then the third and fifth bits of each byte is moved to the beginning of the file according to the scheme shown in Figure 1.

### 3 Searching patterns on $k$ -bit Filtered Files

An input pattern  $P$  of length  $m$  can be viewed as a sequence of bytes  $p_1 p_1 \dots p_m$ , where each  $p_i$ ,  $1 \leq i \leq m$ , is a sequence of eight bits. Given the pattern  $P$  and the set  $R$  of  $k$  indices, the search process begins with decomposing the pattern into two parts as the pattern filter  $PF$  and pattern payload  $PL$  by using the same scheme performed previously on the file.

$PF$  is searched on the filter part of the file, which occupies the initial  $k \cdot n$  bits. In case of possible matches at some bit positions 1 to  $k \cdot n - m + 1$ ,  $PL$  is verified against the corresponding location in the payload part of the file. If  $PF$  is found to begin at a bit position  $f = k \cdot h + 1$  on the  $k$ -bit filtered file, which means there may be a possible match with the  $(h + 1)^{th}$  character of the original file, then  $PL$  should to be verified on the corresponding bit position  $l = k \cdot n + h \cdot (8 - k) + 1$ . The calculation of  $l$  comes from the fact that the payload part of the file begins after  $k \cdot n$  bit filter part, and to reach the payload of the  $(h + 1)^{th}$  byte, one needs to pass over the payloads of the previous  $h$  characters, which makes a total of  $h \cdot (8 - k)$  bits.

### 4 Experimental Results

Tests have been conducted on natural language texts, plain ASCII DNA sequences, and random sequences with uniform probability distribution. The *enwik8*<sup>1</sup> corpus and Manzini's DNA corpus<sup>2</sup> are the sources of natural language and DNA sequences used in the experiments. Random files are generated via the standard *rand()* command of C with *srand(time())* seeds.

During the experiments an input file is saved in the proposed  $k$ -bit filtered format for  $k = 1, 2, 4$ . Sample patterns of length 5 to 50 are randomly selected from the original file. Each pattern is scanned on the input file by the Boyer-Moore [13] (BM), Sunday's quick search[14] (QS), Lecroq's q-hash [6] (LecN),

<sup>1</sup> The *enwik8.txt* file is the subject of the Hutter Prize compression competition, and can be downloaded from <http://prize.hutter1.net>.

<sup>2</sup> <http://web.unipmn.it/manzini/danacorporus>

Pattern Length	Ordinary Files						$k$ -bit filtered files		
	LecN	BLIM	BOM2	BSOM2	BM	QS	1-bit	2-bit	4-bit
5	0.590	0.265	0.329	0.462	0.256	0.217	0.372	<b>0.143</b>	0.148
10	0.524	0.154	0.211	0.291	0.144	0.137	0.093	<b>0.076</b>	0.092
15	0.202	0.122	0.150	0.202	0.109	0.101	0.062	<b>0.059</b>	0.069
20	0.128	0.101	0.126	0.170	0.094	0.093	<b>0.044</b>	0.049	0.056
25	0.094	0.100	0.107	0.142	0.088	0.087	<b>0.042</b>	0.050	0.049
30	0.077	0.092	0.091	0.117	0.079	0.076	<b>0.033</b>	0.036	0.041
35	0.065	0.056	0.080	0.102	0.070	0.070	0.032	<b>0.031</b>	0.039
40	0.057	0.056	0.074	0.095	0.071	0.072	<b>0.028</b>	<b>0.028</b>	0.035
45	0.052	0.056	0.069	0.085	0.066	0.066	<b>0.024</b>	0.033	0.033
50	0.050	0.057	0.064	0.078	0.068	0.064	0.032	<b>0.025</b>	0.031

(a) Average user times on natural language text of 20 MB

5	0.866	0.689	0.947	1.346	0.973	0.940	0.576	<b>0.210</b>	0.360
10	0.753	0.404	0.497	0.712	0.674	0.753	0.139	<b>0.109</b>	0.198
15	0.289	0.331	0.369	0.512	0.558	0.662	<b>0.085</b>	<b>0.085</b>	0.193
20	0.183	0.253	0.286	0.405	0.536	0.647	<b>0.063</b>	0.070	0.144
25	0.136	0.250	0.240	0.335	0.461	0.589	<b>0.059</b>	0.070	0.146
30	0.110	0.241	0.204	0.280	0.449	0.537	<b>0.048</b>	0.052	0.113
35	0.093	0.177	0.187	0.257	0.524	0.809	0.046	<b>0.045</b>	0.142
40	0.082	0.172	0.169	0.224	0.454	0.618	<b>0.038</b>	0.041	0.109
45	0.076	0.167	0.153	0.206	0.441	0.651	<b>0.035</b>	0.049	0.121
50	0.071	0.166	0.143	0.190	0.432	0.690	0.043	<b>0.036</b>	0.107

(b) Average user times on plain ascii DNA sequence of 30 MB

5	0.883	0.213	0.249	0.392	0.289	0.239	0.524	0.209	<b>0.207</b>
10	0.782	0.125	0.140	0.215	0.156	0.138	0.143	<b>0.113</b>	0.134
15	0.301	0.096	0.106	0.155	0.111	0.101	0.088	<b>0.087</b>	0.099
20	0.190	0.077	0.087	0.124	0.090	0.082	<b>0.067</b>	0.071	0.079
25	0.141	0.074	0.079	0.107	0.080	0.074	<b>0.061</b>	0.072	0.068
30	0.113	0.070	0.072	0.095	0.072	0.069	<b>0.050</b>	0.052	0.058
35	0.096	0.064	0.069	0.088	0.069	0.067	0.049	<b>0.046</b>	0.054
40	0.085	0.064	0.066	0.081	0.067	0.067	<b>0.040</b>	0.042	0.049
45	0.078	0.065	0.064	0.078	0.066	0.065	<b>0.036</b>	0.048	0.046
50	0.074	0.065	0.063	0.074	0.065	0.064	0.046	<b>0.037</b>	0.042

(c) Average user times on random byte sequences of 30 MB

**Fig. 2.** Comparison of pattern matching performance between ordinary files and  $k$ -bit filtered files via average user times measured in milliseconds during the experiments.

backward oracle/suffix oracle matching [15] (BOM2/BSOM2), and Külekci’s bit-parallel BLIM [8] algorithms.

Same patterns are scanned on the *1-bit*, *2-bit* and *4-bit* filtered files of the source files with the proposed scheme. Each sample pattern is searched on files five times, and for each length twenty random patterns are used. The experiment is repeated several times on several same length ordinary files. The user times are measured via the *getrusage* command.

Tests were run on a machine having a 64-bit Intel Xeon processor with 3 GB of memory, and best effort was deployed for the implementation of the algorithms. The compiler used was gcc 4.3.1 on Linux core 2.6.25.9-101.

Figure 2 lists the average of the measurements. Best performances are marked in bold.

Search speed is approximately doubled on  $k$ -bit filtered files for all lengths. It is observed that the gain is more significant on DNA sequences, as on short patterns up to length 20, matching via  $k$ -bit filtering is more than 3 times faster when compared with the best performing classical algorithm included in this study.

Note that as the filtered bit length  $k$  increases, so does the distinguishing power. On the other side, using more bits enlarges the length of the file filter part ( $k \cdot n$  bits), which in turn slows down the pattern filter matching. In this trade-off, the results indicate that  $k = 4$  is a bad choice, and up to length 15, selection of  $k = 2$  seems better. Just one bit filter ( $k = 1$ ) is in general more speedy than the best resulting algorithm, except the very short pattern cases on natural language and random texts.

## 5 Conclusion and Future Work

This study focused on exploring an alternative way of storing files for fast offline pattern matching. The proposed  $k$ -bit filter file format aims to create an effective filter over a file from the most informative  $k$  bits of each character occurring in the file. Searching of a pattern in the file is performed by first extracting the filter of the pattern and locating it in the filter part of the target file followed by the verification process on matching positions.

Experiments conducted on natural language, plain ASCII DNA sequence, and random byte texts indicated that exact matching performance is doubled when files are stored in the  $k$ -bit filtered format, even for  $k = 1$ . Selecting  $k = 2$  causes an improvement on short patterns as expected since the distinguishing power is incremented by more bits, but setting  $k = 4$  worsens the performance. It is also observed that highest gain is obtained on 1-bit filtered DNA sequences, especially on patterns shorter than 20 bases.

Selection of the most informative bits is done according to compression ratios of individual bit streams. However, it is also possible to select the indices of the  $k$  bits via some other techniques, such as simply counting the 0/1 ratio, or measuring the variances of the  $B_x$  sequences, or even choosing  $k$  random indices. Another dimension is to define bit indices for each character individually instead

of fixing these positions for all the characters. Further studies on the topic is planned to include these opportunities.

## References

1. Apostolico, A., Galil, Z., eds.: Pattern Matching Algorithms. Oxford University Press (1997)
2. Charras, C., Lecroq, T.: Handbook of exact string matching algorithms. King's Collage Publications (2004)
3. Crochemore, M., Rytter, W.: Jewels of stringology. World Scientific Publishing (2003)
4. Grossi, R., Vitter, J.: Compressed suffix arrays and suffix trees with applications to text indexing and string matching. *SIAM Journal on Computing* **35** (2005) 378–407
5. Wu, S., Manber, U.: Agrep – a fast approximate pattern-matching tool. In: USENIX Winter 1992 Technical Conference. (1992) 153–162
6. Lecroq, T.: Fast exact string matching algorithms. *Information Processing Letters* **102** (2007) 229–235
7. Shannon, C.E.: A mathematical theory of communication. *Bell System Technical Journal* (1948)
8. Külekci, M.O.: A method to overcome computer word size limitation in bit-parallel pattern matching. In: Proceedings of ISAAC'2008. Volume 5369 of Lecture Notes in Computer Science., Gold Coast, Australia, Springer Verlag (2008) 496–506
9. Klein, S.T., Ben-Nissan, M.: Accelerating boyer moore searches on binary texts. In: Proceedings of CIAA. Volume 4783 of LNCS., Springer Verlag (2007) 130–143
10. Kim, J., Kim, E., Park, K.: Fast matching method for dna sequences. In: Proceedings of Combinatorics, Algorithms, Probablistic and Experimental Methodologies. Volume 4614 of LNCS., Springer Verlag (2007) 271–281
11. Faro, S., Lecroq, T.: Efficient pattern matching on binary strings. In: Current Trends in Theory and Practice of Computer Science. (2009) Poster.
12. Faro, S., Lecroq, T.: An efficient matching algorithm for encoded dna sequences and binary strings. In: Proceedings of CPM'09. LNCS (2009)
13. Boyer, R., Moore, J.: A fast string searching algorithm. *Communications of the ACM* **20** (1977) 762–772
14. Sunday, D.: A very fast substring search algorithm. *Communications of the ACM* **33** (1990) 132–142
15. Allauzen, C., Crochemore, M., Raffinot, M.: Factor oracle: A new structure for pattern matching. In: Proceedings of SOFSEM'99. Volume 1725 of LNCS., Springer Verlag (1999) 291–306