SUMMARY Data compression is popularly applied to computer systems and communication systems in order to reduce storage size and communication time, respectively. Since large data are used frequently, string matching for such data takes a long time. If the data are compressed, the time gets much longer because decompression is necessary. Long string matching time makes computer virus scan time longer and gives serious influence to the security of data. From this, CPM (Compression Pattern Matching) methods for several compression methods have been proposed. This paper proposes CPM method for PPM which achieves fast virus scan and improves dependability of the compressed data, where PPM is based on a Markov model, uses a context information, and achieves a better compression ratio than BW transform and Ziv-Lempel coding. The proposed method encodes the context information, which is generated in the compression process, and appends the encoded data at the beginning of the compressed data as a header. The proposed method uses only the header information. Computer simulation says that augmentation of the compression ratio is less than 5 percent if the order of the PPM is less than 5 and the source file size is more than 1 M bytes, where order is the maximum length of the context used in PPM compression. String matching time is independent of the source file size and is very short, less than 0.3 micro seconds in the PC used for the simulation.

key words: dependability, string match, CPM (compression pattern matching), computer virus scan, PPM, context

1. Introduction

Data compression is popularly applied to computer systems and communication systems in order to reduce storage size and communication time, respectively [1]. Since large data are used frequently, string matching for such data takes a long time. If the data are compressed, the time gets much longer because decompression is necessary. Long string matching time makes computer virus scan time longer and has a serious influence on the security of data, because virus scan is executed by searching given virus patterns in the data.

From this, fast string matching techniques for the compressed data without decompression have been proposed. These are called Compression Pattern Matching, or CPM. BW transform suitable for CPM [2], [3] and CPM for Ziv-Lempel coding [4]–[6] have been proposed. However, CPM for Prediction by Partial Match, or PPM [7], has not been proposed yet. PPM is based on a Markov model and uses a context information. It achieves a better compression ratio than BW transform and Ziv-Lempel coding.

This paper proposes CPM method for PPM which achieves fast virus scan and improves dependability of the compressed data. The proposed method encodes the context information, which is generated in the compression process, and appends the encoded data at the beginning of the compressed data as a header. The proposed method uses only the header information.

This paper consists of 5 sections. Section 2 introduces PPM compression. The proposed method is illustrated in Sect. 3. Section 4 demonstrates the evaluation of the proposed method. This paper is concluded by Sect. 5.

2. PPM Compression

PPM compression is based on a Markov model and a symbol in the source data are compressed by using the l or fewer symbols string which is included before the symbol in the source data. Here, l is called the order of the model and the string is regarded as a context information. Each context has its own frequency table which stores the frequency of the following symbols. The symbol in the source data is encoded by entropy coding, such as Huffman coding, arithmetic coding, and range coder, by using the frequency table corresponding to the context. The frequency table has the entry for the special symbol called “escape” symbol in order to show the case that the symbol has not appeared in the context yet. The detail of the encoding procedure is shown in [7]. The following shows the outline of the compression algorithm of PPM.

1. Let A = \{a0, a1, \ldots, aM-1\} be a source alphabet and let D = \{d0, d1, \ldots, d_{n-1}\} be the source data, i.e., \(d_i \in A\) for \(0 \leq i \leq n-1\). Consider \(d_i = a_0\) for \(-l \leq i \leq -1\). Let \(i \leftarrow 0\) and the context information be the empty set.
2. If \(i = n\), terminate the compression procedure. Otherwise, let \(L \leftarrow l\) and read the i-th symbol \(d_i\) in the source data.
3. Consider the string \(C_{i,L} = d_{i-L}d_{i-L+1}\ldots d_{i-1}\). If \(C_{i,L}\) is included in the context information, goto Step 5.
4. Add \(C_{i,L}\) to the context information. Construct a frequency table for the context \(C_{i,L}\), where the frequency of each source alphabet is 0 and that of the “escape” symbol is 1. Goto Step 7.
5. If the frequency of \(d_i\) in the frequency table corresponding to \(C_{i,L}\) is not zero, compress the source symbol \(d_i\) by |


The author is with the Graduate School of Advanced Integration Science, Chiba University, Chiba-shi, 263-8522 Japan.

Manuscript received January 10, 2008.

Copyright © 2008 The Institute of Electronics, Information and Communication Engineers
entropy coding by using the frequency table, increase the frequency value of \( d_i \) in the table by 1, and goto Step 8.

6. Compress the “escape” symbol by entropy coding by using the frequency table, increase the frequency values of both \( d_i \) and “escape” symbol by 1.

7. Let \( L \leftarrow L - 1 \). If \( L > 0 \), goto Step 3. Otherwise, compress the source symbol \( d_i \) by entropy coding by using a frequency table where frequency of each source alphabet is uniform.

8. Let \( i \leftarrow i + 1 \). Goto Step 2.

3. String Matching for PPM Compressed Data

The proposed method appends a header information for string matching at the beginning of the compressed data. The header information consist of the context information without frequency tables. That is, the header information includes all strings of \( l \) or fewer symbols included in the source data. We assume that the data structure of the context information in compression and decompression is tree, because this structure can reduce the memory size and the context search time. Every node excluding the root node of the tree is labeled by source alphabet. The context consists of sequences of the source alphabets corresponding to paths from the root node to other nodes.

Let the distance between the root node and a node be the level of the node. If the distance between the root node and the node \( X \) is \( i \), then \( X \) is the \( i \)-th level node, for example. If a node \( X \) of the \( i \)-th level and a node \( Y \) of the \( (i+1) \)-th level are connected by an edge, then the node \( Y \) is a child of the node \( X \). Let \( A = \{a_0, a_1, \ldots, a_M-1\} \) be a source alphabet.

The following Subsections illustrate encoding and decoding methods of the header information, and string matching method.

3.1 Encoding Method

The proposed encoding method encodes the context tree into binary string. The nodes are examined by breadth first order from the root node and the children of each node are expressed by binary string. The binary string is appended to the compressed data of PPM compression. The following list the detailed encoding procedure.

1. Let \( k \) be the number of the first level nodes, i.e., children of the root node, and let \( S_1 = \{S_{1,0}, S_{1,1}, \ldots, S_{1,k-1}\} \) be set of the first level nodes. Let \( A_1 = \{a_{n_1,0}, a_{n_1,1}, \ldots, a_{n_1,k-1}\} \) be set of labels of the nodes in \( S_1 \). The order of symbols in \( A_1 \) is according to the source alphabet, i.e., \( n_1, p < n_1, q \) for \( 0 < p < q < k \).

Output \( M \) bits string \( B_1 = b_{1,0}b_{1,1} \ldots b_{1,M-1} \) which expresses the children of the root node. That is,

\[
b_{1,m} = \begin{cases} 
1 & \text{if } m \in \{n_{1,0}, n_{1,1}, \ldots, n_{1,k-1}\} \\
0 & \text{if } m \notin \{n_{1,0}, n_{1,1}, \ldots, n_{1,k-1}\}.
\end{cases}
\]

2. Let \( i \leftarrow 2 \).

3. Let \( k_i \) be the number of the \( i \)-th level nodes, and let \( S_i = \{S_{i,0}, S_{i,1}, \ldots, S_{i,k_i-1}\} \) be set of the \( i \)-th level nodes.

4. Execute the following procedure for each node \( S_{i,j} \) \((0 \leq j < k_i)\).
   a. Let \( A_{i,j} = \{a_{n_{i,0}, j}, a_{n_{i,1}, j}, \ldots, a_{n_{i,k_i-1}, j}\} \) be a set of labels of the children nodes of \( S_{i,j} \).
   b. Output \( k \) bits string \( B_{i,j} = b_{i,j,0}b_{i,j,1} \ldots b_{i,j,k-1} \) which expresses the children of \( S_{i,j} \). That is,

\[
b_{i,j,m} = \begin{cases} 
1 & \text{if } n_{i,m} \in \{n_{i,0}, n_{i,1}, \ldots, n_{i,k_i-1}\} \\
0 & \text{if } n_{i,m} \notin \{n_{i,0}, n_{i,1}, \ldots, n_{i,k_i-1}\}.
\end{cases}
\]

5. Let \( i \leftarrow i + 1 \). If \( i \leq l \), goto Step 3.

It is apparent that the length of the header information obtained by the above procedure is \( \left(M + \sum_{i=2}^{l-1} k_{i-1}k_i\right) \) bits.

Figure 1 shows an example of a context tree for \( M = 5 \) symbols source alphabet \( A = \{a, b, c, d, e\} \) and order \( l = 2 \). The first level nodes are encoded into \( B_1 = 11001 \). Since the number of the first level nodes is three, \( k = 3 \). The following nodes are encoded into 3 bits length string. The second level nodes are encoded into three strings \( B_{2,0} = 101, B_{2,1} = 100, \) and \( B_{2,2} = 110 \). The header information is concatenation of these 4 strings, i.e., \( B_1B_{2,0}B_{2,1}B_{2,2} = 11001101100110 \).

3.2 Decoding Method

The following list decoding procedure to obtain the context tree from the header information.

1. Let \( B'_1 \) be the first \( M \) bits of the header and cut \( B'_1 \) from the header. Let \( k' \) be Hamming weight of \( B'_1 \) and let \( N' = \{n'_{1,0}, n'_{1,1}, \ldots, n'_{1,k'-1}\} \) be set of the indices of nonzero elements in \( B'_1 \).

2. Make \( k' \) nodes as children of the root node. Attach the label \( a_{n'_{1,j}} \) to the \( j \)-th child nodes, where \( 0 \leq j < k' \).

3. Let \( i \leftarrow 1 \).

4. Let \( S'_i = \{S'_{i,0}, S'_{i,1}, \ldots, S'_{i,k_i-1}\} \) be set of the \( i \)-th level nodes.

5. Execute the following procedure for each node \( S'_{i,j} \) \((0 \leq
j < k*)

a. Let B'i,j be the first k' bits of the remaining header and cut B'i,j from the remaining header. Let k'i,j be Hamming weight of B'i,j and let N'i,j = \{n'i,j,0, n'i,j,1, \ldots, n'i,j,k'i,j-1\} be set of the indices of nonzero elements of B'i,j.

b. If k'i,j > 0, make k'i,j nodes as children of S'i,j and attach the label an'1,nj,j,m to the m-th children node, where 0 ≤ m < k'i,j.

6. Let i ← i + 1. If i < l, goto Step 4.

3.3 Compression of Header Information

In order to reduce the header size, the header information is compressed. That is, binary strings B1 and B'i,j are encoded by run length coding [8] before output in encoding. In decoding, they are decoded at first after they are read from the header.

3.4 String Matching Method

Let T = t0t1t_l be a string of t symbols, where t ≤ l. The string matching is easily implemented by searching the context tree contained by the decoding procedure mentioned above. That is, T is included in the source data if and only if there exist a path of the nodes labeled by t0t1t_l from the root node in the context tree.

As for the string matching for longer strings, that is, the case t > l, we consider the subchains T_i. Let T_i = t_it_i+1t_i+l-1 and 0 ≤ i ≤ t - l. If T is included in the source data, T_i is included in the context information for any i, where 0 ≤ i ≤ t - l. Although there exists a case that T_i is included in the context information for any i and T is not included in the source data, this procedure can be applied to the computer virus scan using long matching patterns. This can indicate the files which have a risk to be infected by a virus.

4. Evaluation

The proposed method is evaluated by using source files provided by “The Canterbury Corpus” [9] and “Pizza & Chili Corpus” [10].

4.1 Compression Ratio

Table 1 lists the compression ratio of the proposed method for order l = 3, 4, 5, where the compression ratio is obtained by

\[
\text{compression ratio} = \frac{\text{compressed data size}}{\text{source data size}} \times 100.
\]

Here, header information is encoded by run length coding with maximum run length L = 15 bits, as mentioned in Sect. 3.3. For comparison, the cases for PPM [7], FM-index [3], and LZ-index [6] are also listed. Here, FM-index and LZ-index are CPM for BW coding and Ziv-Lempel coding, respectively, and the programs provided by [10] are used. The compression ratios of “english.50 MB”, “english.100 MB”, and “english.100 MB” for FM-index cannot be obtained because the program induces “segmantation fault”. This says that the proposed method augments the compression ratio by less than 3 percent for l = 3. For l = 4, the compression ratio augmentation is about 5 percent for files of greater than 1M bytes. This also says that the compression ratio of the proposed method are better than those of FM-index and LZ-index for files of greater than 100 kBytes.

4.2 Encoding and String Matching Time

The encoding time is obtained by computer simulation. The following are specifications of the PC used for the simulation:

- CPU: AMD Opteron 1216 2.4 GHz
- Memory: 2 G bytes
- OS: Linux 2.6.9-55.ELsmp

Table 1 Compression ratio of the proposed method (%).

<table>
<thead>
<tr>
<th>file name</th>
<th>size (bytes)</th>
<th>PPM l=3</th>
<th>PPM l=4</th>
<th>PPM l=5</th>
<th>proposed method l=3</th>
<th>proposed method l=4</th>
<th>proposed method l=5</th>
<th>FM-index l=3</th>
<th>FM-index l=4</th>
<th>FM-index l=5</th>
<th>LZ-index l=3</th>
<th>LZ-index l=4</th>
<th>LZ-index l=5</th>
</tr>
</thead>
<tbody>
<tr>
<td>alice29.txt</td>
<td>152,089</td>
<td>29.24</td>
<td>29.42</td>
<td>30.46</td>
<td>29.24</td>
<td>44.70</td>
<td>51.98</td>
<td>46.75</td>
<td>171.64</td>
<td></td>
<td>50.43</td>
<td>181.00</td>
<td></td>
</tr>
<tr>
<td>ayyoulik.txt</td>
<td>125,179</td>
<td>31.96</td>
<td>32.14</td>
<td>33.07</td>
<td>31.96</td>
<td>49.35</td>
<td>57.64</td>
<td>34.81</td>
<td>125.18</td>
<td></td>
<td>31.76</td>
<td>194.85</td>
<td></td>
</tr>
<tr>
<td>bible.txt</td>
<td>4,047,392</td>
<td>24.68</td>
<td>23.27</td>
<td>24.25</td>
<td>24.68</td>
<td>23.74</td>
<td>24.91</td>
<td>40.55</td>
<td>106.80</td>
<td></td>
<td>29.00</td>
<td>211.61</td>
<td></td>
</tr>
<tr>
<td>cp.html</td>
<td>24,603</td>
<td>29.96</td>
<td>30.02</td>
<td>30.63</td>
<td>29.96</td>
<td>157.15</td>
<td>194.51</td>
<td>37.87</td>
<td>263.80</td>
<td></td>
<td>43.54</td>
<td>166.59</td>
<td></td>
</tr>
<tr>
<td>E.coli</td>
<td>4,638,690</td>
<td>24.41</td>
<td>24.35</td>
<td>24.38</td>
<td>24.41</td>
<td>24.35</td>
<td>24.39</td>
<td>48.70</td>
<td>168.73</td>
<td></td>
<td>34.56</td>
<td>135.18</td>
<td></td>
</tr>
<tr>
<td>fields.c</td>
<td>11,150</td>
<td>28.11</td>
<td>27.97</td>
<td>28.58</td>
<td>28.11</td>
<td>194.23</td>
<td>362.06</td>
<td>46.39</td>
<td>276.72</td>
<td></td>
<td>132.16</td>
<td></td>
<td></td>
</tr>
<tr>
<td>grammar.lisp</td>
<td>3,721</td>
<td>31.49</td>
<td>31.41</td>
<td>32.05</td>
<td>31.49</td>
<td>267.41</td>
<td>472.30</td>
<td></td>
<td></td>
<td></td>
<td>129.99</td>
<td></td>
<td></td>
</tr>
<tr>
<td>lec10.txt</td>
<td>426,754</td>
<td>27.70</td>
<td>27.10</td>
<td>28.68</td>
<td>27.70</td>
<td>33.40</td>
<td>37.54</td>
<td></td>
<td></td>
<td></td>
<td>43.54</td>
<td></td>
<td></td>
</tr>
<tr>
<td>pfrabn12.txt</td>
<td>481,861</td>
<td>30.73</td>
<td>30.73</td>
<td>31.95</td>
<td>30.73</td>
<td>35.84</td>
<td>39.35</td>
<td></td>
<td></td>
<td></td>
<td>48.70</td>
<td></td>
<td></td>
</tr>
<tr>
<td>world192.txt</td>
<td>2,473,400</td>
<td>27.14</td>
<td>28.45</td>
<td>31.03</td>
<td>27.14</td>
<td>29.73</td>
<td>32.76</td>
<td></td>
<td></td>
<td></td>
<td>34.56</td>
<td></td>
<td></td>
</tr>
<tr>
<td>xargs.1</td>
<td>4,227</td>
<td>38.36</td>
<td>38.46</td>
<td>38.74</td>
<td>38.36</td>
<td>317.08</td>
<td>580.06</td>
<td></td>
<td></td>
<td></td>
<td>46.39</td>
<td></td>
<td></td>
</tr>
<tr>
<td>english.50 MB</td>
<td>52,428,800</td>
<td>30.71</td>
<td>30.34</td>
<td>31.72</td>
<td>30.71</td>
<td>30.44</td>
<td>31.86</td>
<td></td>
<td></td>
<td>*</td>
<td>132.16</td>
<td></td>
<td></td>
</tr>
<tr>
<td>english.100 MB</td>
<td>104,857,600</td>
<td>30.95</td>
<td>30.49</td>
<td>31.86</td>
<td>30.95</td>
<td>30.55</td>
<td>31.95</td>
<td></td>
<td></td>
<td>*</td>
<td>129.99</td>
<td></td>
<td></td>
</tr>
<tr>
<td>english.200 MB</td>
<td>209,715,200</td>
<td>30.81</td>
<td>30.23</td>
<td>31.62</td>
<td>30.81</td>
<td>30.26</td>
<td>31.67</td>
<td></td>
<td></td>
<td>*</td>
<td>126.74</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The cases with '*' are those where program cannot execute because of segmentation fault.
Table 2 lists encoding time of the proposed method, where encoding time is divided into PPM compression time and header generation time. Although the CPU is dual-core CPU, only single core is used for simulation because of single-thread programming. This says that time overhead for header generation is less than 5 percent for the files of greater than 1 M bytes.

String matching time of the sub string including in the source files in the proposed method is independent of the source file size. It is apparent that the complexity is $O(Ml)$, where $t \leq l$. Computer simulation says that string matching time for the above files is less than 0.3 micro seconds. This says that the proposed method achieves fast string matching for compressed data.

5. Conclusion

This paper has proposed string matching for PPM compression. String matching is executed by searching the context information, which is appended at the beginning of the compressed data as header information.

Computer simulation says that augmentation of the compression ratio is less than 5 percent if the order, the maximum length of the context, of the PPM is less than 5 and the source file size is more than 1 M bytes. Time overhead for header generation is less than 5 percent if the source file size is more than 1 M bytes. String matching time is independent of the source file size and is very short, less than 0.3 micro seconds.

Future study remains in developing string matching method which can match longer strings than the order of PPM.

Acknowledgement

The authors are grateful to anonymous reviewers for their useful comments. The authors are also grateful to Mr. Tomoya Ebihara and Mr. Kensuke Tai both of Chiba University for their support on computer simulation. This research was partially supported by the Ministry of Education, Science, Sports and Culture, Grant-in-Aid for Scientific Research (C), 19500052, 2007-2008.

References

Masato Kitakami received the B.E. degree in electrical and electronic engineering, the M.E. degree in computer science, and the Dr. Eng. degree all from the Tokyo Institute of Technology, Tokyo, Japan in 1991, 1993, and 1996, respectively. He joined Department of Electrical and Electronic Engineering, Tokyo Institute of Technology in April 1996 and moved to the Department of Information and Image Sciences, Chiba University in December 1999. From April 2001 to March 2003, he is with VLSI Design and Education Canter, the University of Tokyo. He has been with Graduate School of Advanced Integration Science, Chiba University since April 2007 and is now associate professor. Dr. Kitakami received the Young Engineer Award from the IEICE in 1999. His research interests include error control coding, dependable parallel/distributed systems, error control in data compression. He is a member of the IEEE.

Toshihiro Okura received the B.E. degree in information and image science and the M.E. degree in intelligence information science all from Chiba University, Chiba, Japan in 2005 and 2007, respectively. He joined Kawasaki Microelectronics, Inc. in April 2007 and is now LSI CAD engineer. His research interests include LSI design system optimization.