An Effective GML Documents Compressor

Jihong GUAN†a, Nonmember, Shuigeng ZHOU††b, Member, and Yan CHEN†††, Nonmember

SUMMARY As GML is becoming the de facto standard for geographic data storage, transmission and exchange, more and more geographic data exists in GML format. In applications, GML documents are usually very large in size because they contain a large number of verbose markup tags and a large amount of spatial coordinate data. In order to speedup data transmission and reduce network cost, it is essential to develop effective and efficient GML compression tools. Although GML is a special case of XML, current XML compressors are not effective if directly applied to GML, because these compressors have been designed for general XML data. In this paper, we propose GPress, a compressor for effectively compressing GML documents. To the best of our knowledge, GPress is the first compressor specifically for GML documents compression. GPress exploits the unique characteristics of GML documents to achieve good performance. Extensive experiments over real-world GML documents show that GPress evidently outperforms XMill (one of the best existing XML compressors) in compression ratio, while its compression efficiency is comparable to the existing XML compressors.

key words: Web-based GIS, geospatial data, GML, compression, algorithm, performance evaluation

1. Introduction

To cope with the problems of interoperability and heterogeneous integration among GISs and their applications, the Open Geospatial Consortium (OGC) published the Geography Markup Language (simply GML) specification [1], which offers a vendor-neutral framework for the definition of geo-spatial application schemas and objects. Nowadays, GML is becoming the de facto standard format for geo-spatial data exchange and transmission, more and more geographic information exists in GML format.

GML has many advantages [2], but similar to the use of XML, GML’s flexibility and extensibility also come at a cost: naive use of GML can easily make GML documents be extremely verbose due to the unlimitedly adopting of hierarchical markup and structural information, long elements and attribute names, verbose hyperlinks and namespaces, and sharply storage-volume expanding caused by spatial data (coordinates) transformation from compact binary format to text string. Such redundancy consequently incurs substantial storage and transmission cost, affects both query processing and data exchange. Therefore, it is particularly important to explore effective compression techniques that can considerably reduce storage and transmission cost of GML data. However, current research on GML mainly deals with the issues of storage [3], [4], query processing [5], [6], and information integration [7], [8]. Little work has done for GML compression.

Considering that GML is a special case of XML, a straightforward way of GML compression is to apply the existing XML compressors directly to GML documents. In recent years, a number of XML compressors have been developed, including XMill [9], XMLPPM [10], XGrind [11], XPress [12], XQzip [13], etc. However, these compressors have been designed for general-purpose XML documents (mainly text-rich documents). Quite different from text-rich XML documents, GML documents are data-rich, they contain not only a lot of verbose structural information, but also a large volume of spatial information, i.e., spatial coordinates. We can not expect optimal performance by simply applying general-purpose XML compressors. It is necessary to propose GML specific compressors by exploiting the unique characteristics of GML documents to achieve better compression performance.

In this paper, we explore compression technique for GML documents, and propose a GML specific compressor. We call this compressor GPress, which is the abbreviation of “GML compPressor”. To the best of our knowledge, GPress is the first compressor developed specifically for GML documents. Major contributions of this paper are as follows:

- An approach specifically for GML documents compression is proposed, which exploits the features of GML documents to achieve better compression performance than the existing XML compressors.
- The proposed approach is implemented and a GML specific compressor (i.e. GPress) is built.
- Extensive experiments over real-world GML documents are conducted to evaluate the new compressor, which demonstrate its performance advantage.

The rest of this paper is organized as follows. Section 2 presents related work of GML compression. Section 3 introduces our GPress method. Section 4 gives experimental evaluation on GPress. Finally, Sect. 5 concludes the paper and highlights the future work.
2. Related Work

Currently, we have not seen any published research on GML documents compression. However, on one hand, as GML is a special case of XML for GIS applications, we will survey XML compression techniques. On the other hand, in the literature, there exist some compression approaches for raster and vector maps, so we will also give an overview of these approaches.

2.1 XML Documents Compression

Up to now, a number of XML compression methods or systems have been reported, including XMill [9], XMLPPM [10], XGrind [11], XPRESS [12], XQzip [13], XQueC [14], and XComp [15].

XMill was developed by H. Liefke and D. Suciu [9]. It is based on a regrouping strategy that groups XML text strings with respect to their meaning and exploits similarities between those text strings for compression. The compression principles of XMill are: compressing structure and data separately, data values are grouped with regard to meaning, and compressed by corresponding semantic compressors. XMill typically achieves about twice the compression rate of conventional text compressors like gzip without sacrificing speed. However, XMill still has some disadvantages [16], such as unqueriable compressed output and human intervention required for specifying containers, etc.

XMLPPM was proposed by J. Cheney [10], it uses a technique called Multiplexed Hierarchical Modeling (MHM) that is based on SAX encoding and Prediction by Partial Match (PPM) encoding. In XMLPPM, an input XML document is modelled as a stream of SAX events. The tokens in the SAX event stream are then processed by a set of four PPM coding models corresponding to different syntactic contexts. Although XMLPPM can achieve good compression ratio (sometimes even better than XMill), its efficiency is un-satisfactory.

XGrind was introduced by P. M. Tolani and J. R. Haritsa [11]. It does compression by two main techniques: metadata compression and enum-type attribute value compression, and retains the structure of the original XML document at the same time. Experimental results indicate that its compression ratio improves as the number of enumerated attributes increases. It has lower compression ratio than XMill, but it is a queriable XML compressor that supports querying the compressed document, especially suitable for resource limited computing devices like palmtops.

XPRESS was proposed by J.-K. Min et al [12]. It supports direct and efficient evaluations of queries on compressed data, and adopts a novel encoding method called reverse arithmetic encoding, which is an efficient path encoding method that yields fewer overheads of partial decompression and quicker path evaluation. XPRESS achieves significant improvement on query performance for compressed XML data and reasonable compression ratio.

XQzip was brought forth by J. Cheng and W. Ng [13]. It supports querying compressed XML data by imposing an indexing structure called Structure Index Tree (SIT) that can be used to remove the duplicate structures. XQzip compresses XML data into a sequence of blocks which can be decompressed individually to achieve a good performance. In addition, XQzip supports a wide scope of XPath queries such as multiple, deeply nested predicates and aggregation.

XQueC was presented as a demo by A. Arion et al at VLDB in 2003 [14]. XQueC compresses each data item individually that may result in compression ratio degradation. One important feature of XQueC is that it supports efficient evaluation of XQuery by using a variety of structure information, such as dataguides, structure tree and other indexes. However, the auxiliary structures and the pointers to the data items compressed individually may cause huge space overhead.

XComp was developed by W. Li [15]. With similar compression principles to XMill, XComp takes advantage of the self-describing feature of XML to separate structure from data based on the semantics. XComp also employs a parse-compress architecture that allows parsing and compressing to be done in parallel when the document can not fit in one memory window.

All these XML compressors mentioned above are designed mainly for general XML documents compression. Certainly they can be applied to GML documents compression. Certainly they can be applied to GML documents compression, but we can not expect that the performance will be satisfactory.

2.2 Raster and Vector Maps Compression

Raster map compression methods generally exploit a raster matrix to obtain a concise representation, while main vector map compression methods include line simplification [17], [18], chain coding [19], dictionary based compression [20], and clustering based compression [21].

Line simplification [17], [18] and chain coding [19] were proposed for paper maps and other line-drawings containing curves. These methods approximate curves by a sequence of straight lines, which include vertical, horizontal, diagonal etc. Line-simplification and chain-codes often eliminate original points and add new points as a side-effect. A typical chain-coding-based algorithm FHM (Fibonacci, Huffman, and Markov) was designed for signature compression [19], [22]. The dictionary approach used in the FHM [20] algorithm can be used for compressing digital vector maps. Dictionary-based algorithms build a dictionary to match dictionary entries to line segments. When a dataset is decompressed, the dictionary is searched to find the data associated with the dictionary indices. Recently, Shenkhar et al proposed the clustering-based compression method for vector map [21], [23]. This method adapts a dictionary to a given dataset, which can achieve better approximations of the original map data, so lead to lower error.
3. The GPress Method

In this section, we will first indicate the differences between GML documents and XML documents, then introduce the compression principles of GPress, the architecture of GPress and the technical details of float data compression employed in GPress. Finally, we will give a rough analysis on the reason why GPress works well with GML documents.

3.1 What Makes GML Different from XML?

As in XML documents, GML documents contain large amount of verbose information, such as hierarchical markup and structure, long descriptive elements and attribute names, hyperlinks and namespaces, etc. In addition to that, GML documents also hold large volume of geometric coordinates data, which does not exist in general XML documents. Considering that in real applications the resolution requirement for geographic location is usually very high, often six to eight or even more digits are required for the fractal part of the spatial coordinate values, so the effective digits in total for one coordinate value may be beyond ten. What’s more, a GML document is simply a text file, and the text data is expressed by character string. For a spatial coordinate value, each of its effective digits needs one byte. Hence, it is quite often that a spatial data item in a GML document requires more than 10 or even 20 bytes for storage. On the contrary, when digitizing a map in practice, the geographic data generated is in fact in accord with some kind of strict direction, for example, from left to right or from top to bottom, etc. Hence the series of coordinate values of the same object is monotonous or segment-wise monotonous. Such monotonicity may be exploited in compression. However, when the coordinates are encoded in GML format as actually some text strings, the correlation will be dampened, unless they are explored intentionally.

3.2 The Compression Principles

GPress adopts three compression principles: separating document structure from data items, grouping data items based on their semantics, and compressing coordinate data separately from non-coordinate data.

3.2.1 Separating Structure from Data Items

The structure of a GML document refers to element and attribute names, while data refers to the data items or values (mainly spatial coordinates). In GPress, we extract the structure of GML document, and use a set of integers to keep the positions of tags and data items. 0 to 8 are used to encode the positions of special tokens in the structure, such as the end tag, data item, CDATA, PI, comment, etc. When encoding the structure information, we use integer "8" to represent the start position of coordinates.

For spatial data, no matter the "<coord>...<endcoord>" or the "<coordinates>...<endcoordinates>" are used, the parser will keep the start position of spatial coordinates, and put an integer "8" to the structure container. For coordinates represented by "<coord>...<endcoord>", since its structure is fixed, then the inner structure is neglected, and the coordinates are extracted directly to the corresponding containers and the number of coordinate pairs is kept. The coordinates represented by "<coordinates>...<endcoordinates>" are handled similarly.

3.2.2 Grouping Data Items Based on Their Semantics

The second principle is semantically grouping data items according to the self-describing property of GML documents. Usually the name of an element or an attribute in a GML document implies the type of data that the element or attribute contains. With high probability, data items that have the same element name or attribute name are semantically related. The purpose of the second principle is to put the data items with similar properties together in the same container according to their semantics to achieve better compression. In our GPress compressor, we use similar data grouping method employed by the existing XML compressor XMill.

3.2.3 Compressing Spatial and Non-spatial Data Separately

This is the distinct principle employed in GPress. In GPress, specific containers are used to hold the spatial data (coordinates) and a specific float data delta compression algorithm is developed for the spatial data.

The coordinates sequence of point, line and polygon objects in a vector map represented by GML are usually mono-tonous or segment-wise monotonous. Therefor the difference between the adjacent coordinates is small. And when digitizing a map in practice, the geographic data generated is in fact in accord with some kind of strict direction, for example, from left to right or from top to bottom, etc. Hence the series of coordinate values of the same object is monotonous or segment-wise monotonous. We can observe this in the sample GML document segment shown in Fig. 1, which represents a line-type object.

From Fig. 1, we can see that the coordinates of a line string are monotonous, and adjacent coordinates’ values are close to each other. So we can adopt the delta compression method[24] for spatial data in GML documents. In Sect. 3.4, we will give the details of spatial data compression in GML documents.
3.3 The Architecture

Figure 2 is the architecture of GPress. The semantic compressors in GPress are adopted from XMill. Compared with the XMill framework, a distinct feature with our GPress compressor is that it has a float-point data compression component, which consists of a float-point data compressor, a X-coordinate container and a Y-coordinate container. GPress works as follows: when a GML document comes, it is first parsed by the SAX parser. The geographic spatial coordinates are extracted from the document, and compressed by float data delta compressor, and then stored in float data containers. Here, the X-coordinates and Y-coordinates are set apart and stored in different containers, i.e. X-coordinate container and Y-coordinate container. The document structure is extracted from the document and stored in the structure container, while the other data are analyzed and semantically grouped to different semantic compressors and corresponding data containers. After the first round compression (semantic compression and float data compression), all data separated from the GML document is further compressed by the gzip compressor, and eventually merged to generate the output file.

3.4 Coordinates Data Compression

Note that spatial data compression is the core functionality of GPress, here we will present the details of coordinates data compression. Considering that coordinates data is float data, we use delta compression method to do coordinates compression. Here, we use coordinates data and float data interchangeably.

We use two memory data structures to store the current float data item (waiting for compressing) and the last float data item (or the preceding float data item that has just been compressed), which are shown in Fig. 3. The structure CurrentFloat is used to hold the current float data item, in which isneg indicates whether the current float data item is positive (with its value=1) or negative (with its value=0), point and length mean the position of radix point and the length of the float data item. The float data item is stored in value as a character array. The last float data item is kept in the structure LastFloat. Here, using one bit to save the sign of the last float item, and 7 bits to save its radix-point position. Furthermore, we have a storage structure for saving the compressed result, which is shown in Fig. 4. The storage structure consists of three parts: two bytes and a length-variable array. For the first byte, fig1 (the 7th bit b7) is used to indicate whether the current float data item and the last float data item have similar sign and similar radix-point position. If these two floats have similar sign and radix-point position, then fig1 is set to 0, otherwise 1. flg2 (the 6th bit b6) is used to store the information indicating whether the value offset between the current float and the last float is positive or not. Here we define the offset between the current float and the last float as the result of (the value of current float – the value of the last float). If the current float ≥ the last float, flg2 is set to 1, otherwise 0.
The remaining six bits are used to store the length of the offset value. In the second byte, \texttt{flg3} (the 7th bit \texttt{b7}) is used to indicate the sign of the current float data item. If the current float is negative, \texttt{flg3}=1, otherwise 0. The rest seven bits are used to hold the radix-point position of the current float. The third part of the storage structure is a length-variable array, used to hold the current float’s offset value. Its length is determined by the value stored in the six low bits (\texttt{b5} to \texttt{b0}) of the storage structure’s first byte.

The process of float data compression in GPress is as follows.

- **Step 1:** Read a new coordinate-data item (if it is the first float data item of the GML document, it will be directly saved into the output file without the following steps, then put its corresponding information into the structure \textit{Last Float}, and return to the start of Step 1), take it as the current float data item, extract the corresponding information and put them into the structure \textit{Current Float}.

- **Step 2:** Compare the current float data item and the last one. If the two floats have similar sign and radix point position, then set \texttt{flg1} to \texttt{0}, and we need not to store the second byte of the storage structure to the compression result (i.e., the current float’s sign and radix point position are not saved because they are similar to that of the last float); Otherwise, set \texttt{flg1} to \texttt{1}, and keep the current float’s sign and radix point position information to the compressed result.

- **Step 3:** Compute the offset between the current float and the last one: subtract the last float’s value from the current float’s value to get the offset. If offset > 0, set \texttt{flg2} to 1, otherwise 0.

- **Step 4:** Store the compression result of the current float. That is, to store the information in the storage structure into the compression result file. If \texttt{flg1} is \texttt{0}, we need not store the second byte of the storage structure; Otherwise, all information in the storage structure should be output sequentially into the compressed result file (i.e., the output file). After that, the current float data item turns to be the last float data item, and update the structure \textit{Last Float}, then return to Step 1.

### 3.5 Analysis

Here we give a rough analysis to explain why GPress works effectively with GML documents. From the process of GPress given in Sect. 3.4, we can see that given a float data item \(f\) after the float compression process of GPress, we get \(f_c\). The size of \(f_c\) is at most \((2 + \text{length of offset})\) bytes. If \(f\) has similar sign and radix point position to the preceding data item, the second byte in the storage structure will be neglected, so the size of \(f_c\) will be \((1 + \text{length of offset})\) bytes.

Considering that the proximity of spatial objects in a local area is pervasive, \(i.e.,\) their coordinates are close to each other, and show the property of monotonicity or segment-wise monotonicity. This means that the coordinates have almost similar sign and radix point position, and the offset value is very small, \(i.e.,\) a small \text{length of offset}. In such a situation, \text{length of offset} is quite smaller than the length of the original coordinate value \(f\). Thus we can considerably reduce the data size after the float compression process.

More concretely, let check the sample GML document segment in Fig. 1. Here, the value of the first X-coordinate \(X_1\) is \(734298.026592448\), and the value of the second coordinate \(X_2\) is \(734314.464864463\). Now we conduct GPress float compression on \(X_2\). Because \(X_1\) and \(X_2\) have similar sign and radix point position, and the offset between them is \(16.438272215\), whose length is 11 (not including the radix point because we have the position information of the radix point). So \(X_2\)'s compression result holds \(1 + 11 = 12\) bytes. However, in GML document, \(X_2\) holds
16 bytes. Therefore, GPress compresses 25% of X2's size.

4. Performance Evaluation

In this section, we will evaluate the performance of the GPress compressor, and compare it with XMill, one of the state-of-the-art XML compressors [16].

We implemented the GPress system with VC++, which has been presented as a demo in ICDE 2007 [25]. GPress follows similar interface fashion of the popular compressors, such as Winzip and WinRAR etc. There are three major functions with GPress: Compress, Uncompress, and Find. Compress and Uncompress are for document compression and un-compression respectively, and Find provides a way for the user to check the intermediate results of the compression/uncompression process of GPress. For comparison, in addition to the method proposed in this paper, the XMILL method was also implemented and integrated into GPress. When compressing GML documents, the user can choose GPress or XMill to do the work, or let the system try both methods and report their corresponding results.

4.1 Experimental Setting

We tested the GPress compressor on a PC with a 2.40 GHz CPU and 256 MB memory, and running Windows 2000 OS. We used real-world GML documents of size ranging from 10 KB to 135 MB to test GPress. These documents encode real geographic features of USA, Canada and Mexico, including cities, highways, rivers, lakes, states, etc. The real datasets originate from the sample data files provided by ArcInfo, MapInfo and Oracle with their GIS or spatial database software products. We first transformed these sample data files into GML documents, and then grouped these GML documents into seven sets according to their content: point-object documents (simply pt-documents), line-object documents (simply li-documents), polygon-object documents (simply pl-documents), documents containing point and line objects (simply pl-documents), documents containing point and polygon objects (simply pp-documents), documents containing line and polygon objects (simply lp-documents), and documents containing point, line and polygon objects (simply plp-documents).

Here, point-object documents mean that these documents contain only point objects, such as cities, while line-object documents and polygon-object documents encode only line objects (e.g. highways and rivers) and polygon objects (e.g. lakes and states) respectively. So a plp-document encodes information of point objects and line objects as well as polygon objects. There are two reasons that drive us to organize the testing documents in such a way. On one hand, in GIS applications, each map is composed of a number of map layers, each of which corresponds to a kind of geographic features: point objects, line objects and polygon objects etc. On the other hand, by testing on different GML documents corresponding to different spatial features or their combinations, we can get a more complete performance evaluation.

<table>
<thead>
<tr>
<th>Documents</th>
<th>No. of docs</th>
<th>Min-size</th>
<th>Max-size</th>
</tr>
</thead>
<tbody>
<tr>
<td>pt-documents</td>
<td>5</td>
<td>16.1 KB</td>
<td>1.78 MB</td>
</tr>
<tr>
<td>li-documents</td>
<td>15</td>
<td>74.8 KB</td>
<td>7.71 MB</td>
</tr>
<tr>
<td>pg-documents</td>
<td>12</td>
<td>10.3 KB</td>
<td>23.90 MB</td>
</tr>
<tr>
<td>pl-documents</td>
<td>11</td>
<td>91.4 KB</td>
<td>15.07 MB</td>
</tr>
<tr>
<td>pp-documents</td>
<td>11</td>
<td>26.9 KB</td>
<td>42.08 MB</td>
</tr>
<tr>
<td>lp-documents</td>
<td>12</td>
<td>85.1 KB</td>
<td>66.54 MB</td>
</tr>
<tr>
<td>plp-documents</td>
<td>11</td>
<td>101.7 KB</td>
<td>134.82 MB</td>
</tr>
</tbody>
</table>

Table 1 Real-world documents for GPress testing.

On GPress. Statistics of the seven groups of testing documents is given in Table 1. Here, the 2nd column means the number of documents contained in a documents group, and the 3rd/4th column indicates the minimum/maximum size of documents in a group.

We tested the seven documents groups separately. And for each documents group, Compression ratio (simply CR) and compression time (simply CT) were measured for each document with both GPress and XMill. We use the “bits per byte” CR definition:

\[ CR = \frac{\text{size of (compressed file)}}{\text{size of (original file)}} \times 8 \text{bits/byte}. \]

Obviously, a better performed compressor achieves a relatively lower value. Later we can see that GPress has lower CR value than XMill, which means that our GPress boasts of better compression performance.

4.2 Experimental Results

The experimental results are shown in Fig. 5, where the vertical and horizontal axes correspond to compression ratio or compression time and document size respectively. The diagrams from Fig. 5 (a) to Fig. 5 (g) show the compression ratio results of the seven GML documents groups in Table 1; Fig. 5 (h) compares the compression times of GPress and XMill. For saving space, we give only compression time results of point/line/polygon-object documents (or plp-documents), because this documents group contains more large-documents. The largest document we have tested is in this group, whose size is about 135 MB.

From Fig. 5, we can see that:

- GPress outperforms XMill for all testing documents in compression ratio. So as a GML specific compressor, GPress is successful and satisfactory.
- The compression time is linear to the document size for both XMill and GPress. Nevertheless, GPress costs a little more time than XMill does. The reason is that GPress need additional time to separate coordinate and non-coordinate data. But the difference is not evident. For a 135 Mb GML document (the largest we have tested), XMill spends 23.7 sec., while GPress costs about 27.8 sec., 13% more than XMill consumes.

To carefully check GPress' advantage of compression ratio, we further computed the CR improvement ratio (or simply CRIR) of GPress over XMill. Given a testing document, we define CRIR as follows.
For each documents group, we first computed the CRIR value of each document, and then computed the averaged CRIR over all documents in the group. The results are given in Table 2. From Table 2, we can see that except for the point-object documents, GPress outperforms XMill by nearly 20% in compression ratio for all other six documents groups. Even for the point-object documents, GPress' CRIR is 10%.

\[
CRIR = \frac{CR \text{ of XMill} - CR \text{ of GPress}}{CR \text{ of XMill}} \times 100\% \quad (2)
\]

<table>
<thead>
<tr>
<th>Documents group</th>
<th>Averaged CRIR(%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>pt-documents</td>
<td>10</td>
</tr>
<tr>
<td>li-documents</td>
<td>21</td>
</tr>
<tr>
<td>pg-documents</td>
<td>22.3</td>
</tr>
<tr>
<td>pl-documents</td>
<td>20.4</td>
</tr>
<tr>
<td>pp-documents</td>
<td>20.4</td>
</tr>
<tr>
<td>ip-documents</td>
<td>18</td>
</tr>
<tr>
<td>plp-documents</td>
<td>18.9</td>
</tr>
</tbody>
</table>
Roughly, the reason why GPress has smaller CRIR on point-object documents lies in two aspects: 1) point-object documents have relatively less coordinates than the documents in the other six documents groups, which contain more complex objects than points; 2) Point objects’ coordinates have less intrinsic correlation than line/polygon objects’ coordinates.

5. Conclusion

GML documents contain a large amount of verbose markup and structural information, hyperlinks and namespaces, and a large volume of spatial data, which incurs substantial redundancy for data storage and considerable network cost for transmission. In this paper, we propose an effective GML compressor GPress for GML documents compression based on three principles: separating structure and data items, grouping data items based on their semantics, and compressing spatial data and non-spatial data separately. The compression method used for float coordinates takes advantage of the geographic correlation of spatial data to achieve good compression performance.

The current GPress compressor aims at reducing storage and transmission cost, it does not support query processing. Because spatial queries are much more complex to process than the ordinary database queries, existing queriable XML compressors can not be directly applied to GML, one of our future tasks is to make GPress a query-aware compressor. Considering that there are still much space for further improvement on GPress’ performance, our another future task is to explore more powerful versions of GPress in both compression ratio and compression efficiency. One direction is to employ the PPM method and the semantic characteristics of GML documents to achieve better compression performance.

Acknowledgments

This research was supported by the National Basic Research Program of China under grant No. 2007CB310806, the National Natural Science Foundation of China under Grant Nos. 60496327, 60573183, and 90612007, and the Program for New Century Excellent Talents in University of China (NCET-06-0376).

References

Jihong Guan was born in 1969. She is now a full professor in Department of Computer Science and Technology, Tongji University, Shanghai, China. Her research areas cover spatial databases and data mining, GIS and GML data management, and P2P computing. She received her Bachelor Degree in Computer Science from Huazhong Normal University, Wuhan, China in 1991, and her Master Degree in Computer Science and PhD in Photogrammetric and Remote Sensing from Wuhan University, Wuhan, China in 1998 and 2002 respectively.

Shuigeng Zhou was born in 1966. He received his Bachelor's degree from Huazhong University of Science and Technology in 1988, his Master's degree from University of Electronic Science and Technology of China in 1991, and his PhD from Fudan University in 2000. He served in Shanghai Academy of Spaceflight Technology from 1991 to 1997, as an engineer and later a senior engineer (since August 1995) respectively. He was a postdoctoral researcher in State Key Lab of Software Engineering, Wuhan University from 2000 to 2002. Currently, he is a full professor in Department of Computer Science and Engineering, Fudan University. His research interests include database and data mining, information retrieval, P2P and sensor networks.

Yan Chen was born in 1981. She is now a Master student majoring in Computer Science, in the School of Computer, Wuhan University, China. Her current research area is GML management and compression.