Chapter 10
Compressing and Vague Querying (XCVQ) Design

Badya Al-Hamadani
University of Huddersfield, UK

Joan Lu
University of Huddersfield, UK

ABSTRACT
As shown in the literature review from the previous chapter, there are a good number of studies in the field of compressing XML documents and querying the compressed version without the need to fully decompress. However, vague queries, which are one of the most important query types, have been processed to retrieve information from raw XML documents and not from compressed ones. Depending on the SDM, the design of the complete system should be made, followed by its implementation which can be seen in Appendix B in chapter 12. This chapter illustrates the design architecture of the XCVQ (an XML Compressing and Vague Querying) which has the ability to compress the XML documents and use the compressed files in order to retrieve information according to vague queries. It starts with the main architecture of the system followed by the design of each of its parts, namely XCVQ’s compressor, decompressor, and the query processor.

1.1 SYSTEM ARCHITECTURE
As illustrated in Figure 1, the XCVQ system consists of two main stages. The first is designing a new XML compression technique which converts the normal XML documents to a compressed version. The second is designing a retrieving technique that processes the XPath vague queries in order to retrieve the relevant information from the compressed document accordingly.

The design of the XCVQ does not rely on the XML Schema or the DTD of the document. This is due to several reasons:
The main purpose of designing XCVQ is to process vague queries which are usually written, as illustrated in a previous section, by inexperienced users who may not want to have another technology linked with their documents.

Even if the schema for a document exists, it could not have been accessible to the user.

Since the main purpose of any compressor is to reduce the storage memory and the transition bandwidth, XCVQ saves the amount of memory required to store the schema.

As illustrated in the design of the XCVQ, all the compressed XML documents are stored in a repository which is going to be used in the retrieving process. To the best of our knowledge, XCVQ may well be considered to be the first retrieving technique that has the ability to retrieve information from more than one XML document without requiring the pre-specification of the documents needed to be retrieved and without dependence on the document’s schema. This approach helps users retrieve more relative information no matter which documents contain this information.

The complete design of the XCVQ is illustrated in Figure 2.

The following sections demonstrate the design of each part of the system starting with XCVQ-Compressor (XCVQ-C), passing by XCVQ-Decompressor (XCVQ-D), and ending with XCVQ-Query Processor (XCVQ-QP).

1.2 XCVQ-C DESIGN

XCVQ-C compressor takes an XML document as the input and creates the compressed version from this document by passing through several steps. An example in Figure 3 from (W3Schools.com, 2006b) will be used in the following sections in order to simplify the exact process of each step.

1.2.1 Creating the Structured-Tree and its Abridgment

As illustrated in Figure 2, the first step in compressing the XML document is to create the structured-tree using the SAX parser. This parser scans the XML documents only once and it cached several events such as start-document, start-element, end-element, and end-document. Section 0 contains more details about this parser and its advantages. During this parsing process the complete path-dictionary was created and separates the data part of the XML document from its structure to be abridged to the structured-tree. The structured-tree for the running example is shown in Figure 4. The data under each root-leaf path are stored in containers linked to that path.

Each data item is accompanied with a number \( ID_{order} \) that represents the order of this item within the document (the number between the brackets in Figure 4). \( ID_{order} \) counts each start element, data value, attribute name, attribute value and end element. According to this number, each node is uniquely identified for the purposes of decompression process and in the querying process. Previous XML compressors used two