A MODEL FOR QUERYING SEMISTRUCTURED DATA THROUGH THE EXPLOITATION OF REGULAR SUB-STRUCTURES

Mathias Neumüller and John N. Wilson
Department of Computer and Information Science, University of Strathclyde, Glasgow G1 1XH

Key words to describe the work: Semistructured Data, Query Processing, Structure Mining, Data Optimisation

Key Results: Multiple techniques for querying semistructured data are based on the principle of identifying structured sub-sets of the data, which are relevant to a specific class of queries. These can be exposed using appropriate techniques.

How does the work advance the state-of-the-art?: Definition of a common model for query optimisation

Motivation (problems addressed): Efficient querying of semistructured data, especially XML data in web applications

Introduction and Model

Much research has been undertaken in order to speed up the processing of semistructured data in general and XML in particular. Many approaches for storage, compression, indexing and querying exist, e.g. [1, 2]. We do not present yet another such algorithm but a unifying model in which these algorithm can be understood. The key idea behind this research is the assumption, that most practical queries are based on a particular pattern of data that can be deduced from the query and which can then be captured using a regular structure amendable to efficient processing techniques.

To aid understanding, we divide the problem of query optimisation into five distinct steps, which are presented in Figure 1 and described below:

1. **Design of an index for a particular query class:** Given a particular query pattern or class, indices can be devised, which support easy evaluation. If the index is covering, we can even dispose of the original data and resolve the query entirely using against the index.
2. **Calculation of a clustering based on the index:** The data, if seen as a data graph, can now be clustered into equivalence classes based on the index. The resulting graph has only one node per equivalence class, i.e. combines nodes of the data graph, which cannot be distinguished by any query over the index, leading to a simplification of the structure.
3. **Exposing instance specific structure:** This clustering itself exposes typical structures in the data instance, i.e. from all possible structures governed by the query pattern only those are materialised that occur in the specific data instance. This forms an instance specific schema, which can be used for data exploration and query formulation, e.g. for querying by example.
4. **Deriving data organisation:** The structures discovered can also be used to group the data content, i.e. the atomic data associated with the leaf nodes, into syntactically and semantically homogeneous domains. Note that the semantic is indirectly dependant on the class of queries, i.e. the computation to be performed. This results in a physical data representation, which is appropriate with respect to the class of queries and specific data instance.
5. **Generation of query execution plan:** Based on the generated data organisation, specific queries of the general class can be evaluated efficiently. This last step consists of finding algorithm, which makes use of the partial pre-computation, which has been achieved by the previous data reorganisation.

Not all parts of this problem can be described in detail here. We will, however, illustrate the approach using a simple class of queries, in this case simple node queries with data predicates, and also investigate step four and five in further detail for a more general class of queries.

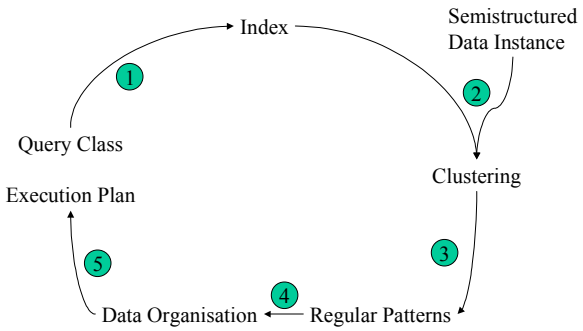

Figure 1: Query optimisation cycle

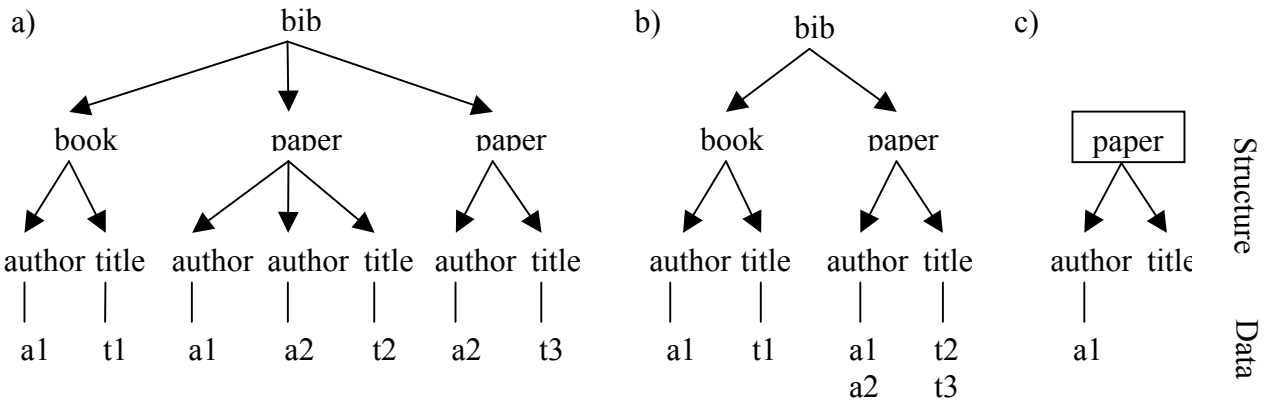

Figure 2: a) Data graph instance, b) forward bisimulation graph, and c) twig query

Simple Value Predicate Queries

For the sake of simplicity we will assume an almost trivial class of queries. We are interested to ask for nodes with a specific name containing a particular piece of atomic information, i.e. the path query pattern *//$label$/text()=$content$*, where *$label$* and *$content$* are variables. We need a two-dimensional index *I(label, content)*, which indexes all nodes with have atomic context. Note that the resulting structural clustering is partial and non-overlapping in this case, i.e. every node of the data graph is mapped to either zero or one index nodes. This simple structure will only show existing label names as domains. Within these domains, only those data content which can be found below such a node in the source, will be stored. The physical data storage in this case can be realised in many ways, most easily as a two-dimensional sorted list. Querying is than as easy as executing a binary search over this list, in which a number of node identities can be stored and returned.

Twig Queries and Branching Path Expressions

We will now show how to use a similar approach on a more complex class of queries, in this case twig queries with value predicates, a subset of branching path expressions as described by Kaushik et al. [2].

Figure 2 shows a simple example. Due to the higher expressiveness of the language, we cannot describe the derivation of a covering index here, but refer to their paper on this topic, which deals with the general query class. Here we will primarily deal with the step four and five described above.

Kaushik et al. describes the generation of an index graph, which is based on bisimilarity of the structure of a graph. Our restriction to branching path expressions means that we can restrict the query operations and thus the index to the downward axis, i.e. base the index on forward bisimilarity alone. Their work and implementation solely deals with structural queries, thus excluding leaf node textual data. We have included this data in dictionaries at leaf node level and implemented a simple top-down query execution strategy. Note that due to the inclusion of leaf data, the index is no longer covering and we must validate the candidate set generated by querying the index graph using the data graph. Because of the used clustering, we will have to look at less data than if we had evaluating the query over the entire data set. Furthermore we know the structure of the candidate set, which can be used for optimisation purposes, like executing the validation in a typed environment.

Conclusions and Outlook

We have presented a model capable of describing query optimisation techniques developed for the processing of semistructured data. We believe that it is general enough to describe a significant number of existing techniques, and have given supporting evidence for two examples. The model is based around the assumption that the core to query optimisation is based on identifying regular structures in a semistructured dataset, which account for most of the data volume.

We have yet to prove under which condition our approach holds, and investigate the performance implications of our thesis. To this end we are working on a flexible experimental system that can be used to test various optimisation techniques in a common framework.

References

1. Buneman, Grohe, and Koch. Path queries on compressed XML. In *Proc. of VLDB*, 2003.
2. Kaushik, Bohannon, et al. Covering Indexes for Branching Path Queries, In *Proc. of SIGMOD*, 2002