INSTITUT FÜR INFORMATIK
Lehr- und Forschungseinheit für
Programmier- und Modellierungssprachen

Oettingenstraße 67 D–80538 München

Ludwig
Maximilians-
Universität
München

Content-Aware DataGuides for Indexing Semi-Structured Data

Felix Weigel

Diplomarbeit

Beginn der Arbeit: 15.09.2002
Abgabe der Arbeit: 05.04.2003
Betreuer: Prof. Dr. François Bry
Dr. Holger Meuss

Erklärung

Hiermit versichere ich, daß ich diese Diplomarbeit selbständig verfaßt habe. Ich habe dazu keine anderen als die angegebenen Quellen und Hilfsmittel verwandt.

München, den 5. April 2003 Felix Weigel

Abstract

This work presents the *Content-Aware DataGuide* (CADG), a new index for tree-shaped Semi-Structured Data. Based on the well-known DataGuide index, its major contribution is the simultaneous use of both structural and textual query criteria during several phases of the retrieval process. Two new pruning techniques for *content-aware path matching* are presented – one based on index node IDs and the other on a novel use of heuristic keyword signatures – which narrow down the search space at evaluation time considerably. This not only avoids needless index navigation, but also some expensive disk operations. Furthermore, *content-aware occurrence fetching* substantially enhances database look-ups. The new index structure is tested extensively with different sets of both manually and automatically generated queries on a variety of document collections, including data from the XML Benchmark Project. Three variations of the CADG are compared to each other and to the original DataGuide. Further techniques for retrieval optimization are tested with all index structures. A *generic query classification scheme* is presented which allows for a systematic comparison and analysis of the performance results (which are also appended for further examination). The experiments prove that the CADG is superior to the DataGuide in virtually all cases, often with a performance gain of 50% and more. Moreover, the new approach scales well to large document collections.

Zusammenfassung

Mit dem *Content-Aware DataGuide* (CADG) präsentiert die vorliegende Arbeit einen neuartigen Index für semistrukturierte Baumdatenbanken. Eine wesentliche Neuerung gegenüber dem bekannten DataGuide-Index, auf dem der CADG basiert, besteht in der kombinierten Auswertung von strukturellen und textuellen Anfragekriterien, d. h. von Suchpfaden und Suchwörtern, in verschiedenen Phasen des Auswertungsprozesses. Es werden zwei neue Verfahren zur suchwortabhängigen Navigation (*content-aware navigation*) im Index vorgestellt, mit deren Hilfe der Suchraum zur Auswertungszeit z. T. erheblich verkleinert werden kann: ein exaktes, auf Knotenbezeichnern basierendes Verfahren sowie ein zweites, das heuristische Suchwortsignaturen auf neue Weise einsetzt. Beide vermeiden nicht nur überflüssige Pfadvergleiche im Index, sondern auch teuere Datenbankzugriffe. Die Integration der strukturellen und textuellen Teile einer Anfrage ermöglicht es dem CADG darüber hinaus, effizientere Anfragen an die darunterliegende Datenbank zu stellen (*content-aware occurrence fetching*). Die neue Indexstruktur ist ausgiebigen Tests unterzogen worden, bei denen große Mengen sowohl händisch als auch automatisch erzeugter Anfragen an verschiedene Baumdatenbanken gestellt werden, etwa aus dem XML Benchmark Project. Hierbei sind drei Varianten des CADG untereinander und mit dem ursprünglichen DataGuide verglichen worden. Alle Indexstrukturen sind außerdem in Kombination mit weiteren Optimierungen getestet worden. Mit Hilfe eines neuen generischen Modells zur Anfrageklassifizierung werden die Testergebnisse systematisch verglichen und ausgewertet. (Sie sind zur weiteren Analyse dem Text beigefügt.) Die Experimente belegen, daß der CADG dem herkömmlichen DataGuide in so gut wie allen Fällen überlegen ist, häufig um 50% oder mehr. Zudem ist der neue Ansatz skalierbar, d. h. auch für große Dokumentsammlungen geeignet.

Acknowledgements

This work would not have been possible in the present form without the help from various people both at the Department of Computer Science and the Centre for Information and Language Processing at the University of Munich (LMU). First of all, I would like to thank my supervisors, Prof. Dr. François Bry and Dr. Holger Meuss, for their most valuable support and expertise in all phases of the project. I have very much enjoyed working with and learning from you, and am looking forward to future co-operation in this or other exciting domains of research. The test retrieval system used for experimental evaluation, on which my implementation is based, is entirely Holger's work. Once again my warmest thanks for your generosity and kindness, reflected also in all the long and fruitful discussions we had. Furthermore I am very grateful to Tim Furche, inexhaustible source of knowledge and advice in various scientific domains, for his generous support. Tim also provided the query generator which I could adapt to my needs. I have often profited from his and Dan Olteanu's scientific insight and friendly contact. Furthermore, I have gratefully received technical assistance with the performance evaluation from the department's system administrator Martin Josko as well as from his colleagues at the Centre for Information and Language Processing. The noun phrase document collection used for the experiments has been compiled by Jürgen Oesterle and Petra Maier-Meyer in an earlier work. The synthetic data were assembled using the generator tool from the XML Benchmark Project. The collection of German cities has been provided by Holger Meuss and his students.

Contents

List of Tables

List of Figures

List of Algorithms

Chapter 1

Introduction

1.1 A new approach to indexing Semi-Structured Data

Since the advent of XML [BPSMM00], the Semi-Structured Data model (see section 2.1) has become widely accepted and is now used in a much broader range of domains than in the early times of SGML [SGM86]. With the growing number of XML-enabled applications, more attention has been paid to efficient indexing and retrieval techniques for Semi-Structured Data. Today's most promising approaches are mostly based on the DataGuide [GW97a], a path index which is combined with a simple inverted file to process combined path and text queries. However, it turns out that when interleaving the two selection criteria content and structure during retrieval, such queries can be evaluated much more efficiently.

The major contribution of this work is an enhanced index structure for tree-shaped Semi-Structured Data, the *Content-Aware DataGuide* (CADG), which makes use of both structural and textual query criteria simultaneously during several phases of the retrieval process. In particular, path matching is carried out in a *content-aware navigation* step which allows to prune irrelevant parts of the index tree. This can be achieved with either exact or heuristic new pruning techniques, which are a substantial improvement on existing approaches (see chapter 6). As a result, less precious evaluation time is wasted in needless index navigation and database accesses. Furthermore, the CADG's integrated *content-aware occurrence fetching* step for both structural and textual hits avoids many expensive disk operations. It also replaces the DataGuide's intersection operations on large occurrence sets with efficient, index-supported database joins.

The second contribution of this work is the extensive and systematic experimental evaluation of the CADG. Three variations of the new index are compared to each other and to the DataGuide for a wide variety of both manually and automatically generated queries on four different document collections, the largest containing more than 500 MB of highly structured data. A *generic query classification scheme* is presented which allows for a systematic comparison and analysis of the numerous performance results. The experiments prove that the CADG is superior to the DataGuide in virtually all cases, with a performance gain of 50% and more. Moreover, the new approach scales well to large document collections. Additionally, four optimization techniques are tested with the different index types to assess their effect on both query evaluation performance and storage requirements. As a result, a bottleneck of the query evaluation algorithm (which is not part of any index structure) is identified, leaving room for future research (see section 7.2).

1.2 Overview of the Thesis

This work is organized as follows. The next chapter deals with some formal and conceptual preliminaries. Most prominently, the DataGuide is introduced, as well as a tree query formalism to be used throughout this work. Chapter 3 presents the Content-Aware DataGuide from a conceptual point of view, including the different variations mentioned above. Chapter 4 describes the implementation accomplished in the course of this work, covering both design issues and details about selected data structures and operations. It also serves as documentation for the test retrieval system. The experimental evaluation discussed above is part of chapter 5. Here the set-up of the different test suites, including various query sets and document collections, is explained in detail, before the experimental results are presented and analyzed. The actual evaluation starts with an overview of the individual test suites and then examines the different index structures and optimizations in detail. (The underlying performance results for all test suites are provided in appendix A.) Chapter 6 reviews related indexing approaches and optimization techniques, contrasting them with the CADG. The thesis concludes with a short summary and outlook for future work in chapter 7. Besides, three appendices provide additional information related to the implementation and evaluation chapters. As just mentioned, the raw experimental results are given in appendix A. Appendix B lists one of the query sets used for evaluation of the CADG. Finally, appendix C documents the retrieval system's command-line interface.

Chapter 2

Preliminaries

Before presenting the Content-Aware DataGuide (CADG) as the main contribution of this work, a concise review of both formal and conceptual preliminaries is in order. After a short discussion of the Semi-Structured Data model, a few notions and notations regarding documents and queries are summarized, which are used throughout this work. In particular, a generic tree query formalism is introduced along with some comments on query evaluation which apply to all index structures considered in the following chapters. Finally, two ground-breaking indexing techniques known from the domains of Semi-Structured Data and Information Retrieval, respectively, are reviewed, both serving as a basis of the CADG. Readers already familiar with these well-known approaches may choose to skip the last two sections of this chapter, as well as the introductory motivation.

2.1 Relational vs. Semi-Structured Data

Since the seventies, the *relational data model* has become the predominant data model in both the commercial and the scientific domain. Over the years, numerous relational database systems with ever more sophisticated index structures have been developed and successfully used in all kinds of application. However, based on flat tables described by a rigid schema, relational data are not particularly suitable for complex hierarchical data with an irregular structure, including many marked-up documents containing natural language such as, most prominently, the enourmous amount of pages available in the World-Wide Web. For this kind of data, the *Semi-Structured Data model*[1] [Abi97, Bun97, Suc98] has been proposed, which gained considerable momentum with the advent of XML [BPSMM00] as a serialization formalism in the late nineties, but was applied to the document markup language SGML [SGM86] more than two decades before. Much attention has been paid to the integration of the relational and the Semi-Structured Data model [FK99, MFK+00, STZ+99, WLOT01, SKWW01, SYU99], which shows that the relational data model alone does not meet all requirements for efficient storage and retrieval of Semi-Structured Data.

There are other approaches to modelling data with an inherent, though irregular structure. While early techniques from classic Information Retrieval used to regard human-readable documents as unstructured sequences of words, several data models proposed in the late eighties strive to capture the internal structure of documents [NBY97]. Yet they are often either too restricted in their expressivity, or on the contrary too complex for efficient retrieval. The Semi-Structured Data model can be considered a compromise between expressivity and efficiency. A wealth of applications based on XML (and the earlier SGML), along with implemented formalisms for querying,

[1] To be precise, there are several a couple of approaches to modelling Semi-Structured Data, differing e.g. in whether parent/child relationships and cross-links are conceptually different, whether there may be multiple roots, or whether the order among sibling nodes matters or not. This work favours an approach similar to the graph-based data model for XML presented in [GMW99], which in turn is based on OEM, the *O*bject *E*xchange *M*odel [PGMW95].

indexing, transforming, and rendering these data, prove that it is sufficiently expressive for a broad range of domains, and at the same time amenable to efficient processing.

2.2 Document trees

Figure 2.2 on page 17 depicts a structured document both in its serialized XML form *(a)* and as a *document tree*[2] *(b)*. Although only a single document is shown, the tree usually contains multiple documents from a *document collection*. As can be seen in the figure, the name of any element in *(a)*, appearing in its start and end tag, is used in *(b)* to label the edge leading to the corresponding node in the tree.[3] For instance, both the XML document's root element `<book>` and the edge to the document tree's root node are labelled "book". Each node has a unique *identifier* (ID) consisting of a positive integer, preceded by the ampersand (&) sign. Unlike XML elements nested within the same parent element, sibling nodes in the document tree are considered unordered here.

Some elements of the XML document in figure 2.2 *(a)* contain text fragments, which consist of individual *keywords*. Usually many words in documents are ignored during index creation, either because they lack specific semantics (so-called *function words*, like conjunctions, prepositions, and determinators) or because they occur just too often to be useful as selection criteria. There are different Information Retrieval techniques to determine which words to index, which are beyond the scope of this work.[4] In the following, it is assumed that the documents in a collection only contain selected keywords considered worth indexing. In figure 2.2 *(b)*, they are attached to their respective containing document nodes.

The following concepts are related to the data model described above:

Definition 1 (DOCUMENT PATH) A *document path* is a sequence of document tree nodes such that each node in the path is a child of the preceding node in the sequence, if there is one. For example, in figure 2.2 *(b)*, there is a document path /&0/&3/&7. The document node &7 is said to be reached by this document path, which is therefore also referred to as &7's path in the document tree.

Definition 2 (ABSOLUTE DOCUMENT PATH) An *absolute document path*, also called *absolute path* or *root path* for short, is a document path (see definition 1) whose first node is the root of the document tree. For instance, the document path /&0/&3/&7 in figure 2.2 *(b)* is absolute.

Definition 3 (LABEL PATH) A *label path* is a sequence of labels belonging to the edges of nodes which together form a document path (see definition 1). The order of the labels is induced by the order of the nodes in this document path. For example, the label path /book/chapter/section corresponds to the document path /&0/&3/&7 in figure 2.2 *(b)*.

Definition 4 (LABEL/ID PATH) A *label/ID path* is a label path (see definition 3) whose labels are annotated with the IDs of their respective nodes in the corresponding document path. For example, in figure 2.2 *(b)*, there is a label/ID path $/\mathsf{book}_{\&0}/\mathsf{chapter}_{\&3}/\mathsf{section}_{\&7}$.

Definition 5 (DIRECT CONTAINMENT) A node n_1 *directly contains* another node n_2 if and only if n_2 is a child of n_1. For instance, in figure 2.2 *(b)* the document node &3 contains &4 and &7 directly, but not &8.

Definition 6 (INDIRECT CONTAINMENT) A node n_1 *indirectly contains* another node n_2 if and only if there exists a document path (see definition 1) containing both n_1 and n_2 in which n_1 precedes n_2. Note that this includes direct containment (see definition 5) as a special case. For

[2] As mentioned in section 1.1, this work only covers tree-shaped documents, ignoring data with *node-sharing* (directed acyclic graphs) and *backlinks* (cyclic graphs).
[3] Alternatively, the node itself could be labelled.
[4] For an overview of Information Retrieval techniques, see e.g. [SM83, vR79, Vir].

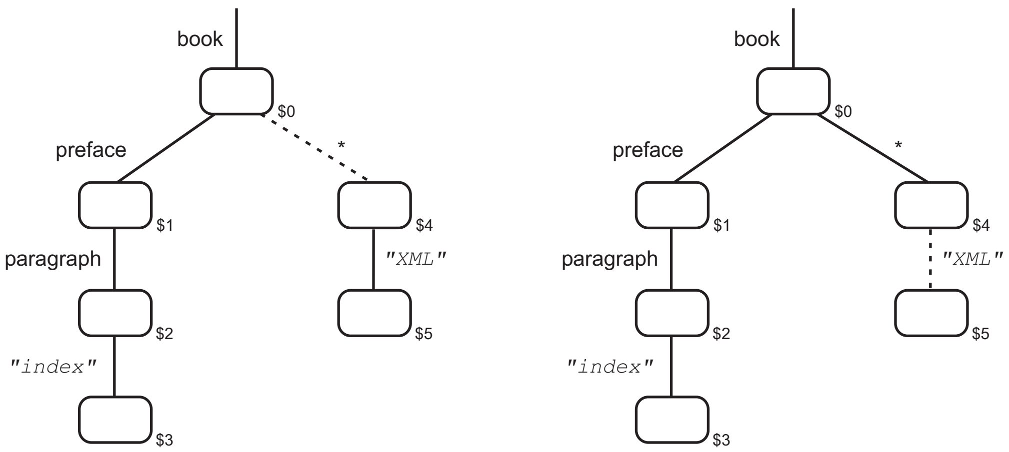
instance, in figure 2.2 *(b)* the document node &3 indirectly contains all document nodes from &4 to &8, which are often referred to as *descendants* of their common *ancestor* &3.

The concept of containment expressed in definitions 5 and 6 is not restricted to document nodes, but can also be applied to their textual content. This becomes even more apparent in data models which represent textual content as *textual nodes* in the document tree, as opposed to *structural nodes* like the ones shown in figure 2.2. For instance, the W3C's XML Information Set [CT01] is such a data model.

2.3 Tree queries

This section deals with the tree query formalism employed throughout this work. It is a simplified version of the one described in [Meu00, MS01, Meu98], which in turn is based on Tree Matching [Kil92]. Being a (albeit indispensable) means to and end rather than the core of this work, the query formalism is only sketched briefly here, on an intuitive basis. For a more formal specification of its semantics, the reader is referred to [Meu00].

Query trees conceptually resemble document trees, as presented in the previous section. The main difference between the two lies in their purpose and their generality. While document trees are a means to describe what a document collection looks like, representing it in a hierarchical and compositional manner, a query tree (like any query expression) specifies which parts of a document tree are to be selected, i.e. localized on disk and returned to the user in a suitable *result data structure*. The process of determining which portions of a document fulfill, or *match*, a given query's selection criteria is called *query evaluation*. Usually it presupposes the existence of index structures for the document collection to be queried. They are created before, at *indexing time*, as opposed to *evaluation time* (or *query time*).



(a) query with soft-edged structural node

(b) query with soft-edged textual node

Figure 2.1: Tree queries

Figure 2.1 presents two sample query trees. Analogously to the document tree depicted in figure 2.2 on page 17, there are two different selection criteria expressed in a tree query, namely *structure* and *content*. Accordingly, query evaluation combines two tasks, *structure matching* (and in particular *path matching*) as well as *content matching*. A query tree's structural selection criteria are represented by its *structural nodes*, i.e. the ones with labels from the document collection. Roughly speaking, a given structural query node is matched by every document node whose path consists of nodes with the same labels as in the query node's path (in the same order, of course). Such document nodes qualify as *structure hits*, which additionally have to match the textual selection criteria to be included in the final result. Edges to a structural query node may bear an undefined label, denoted $*$, which is matched by any document node's label. Unlike document tree edges, however, edges to nodes of a query path are not only distinguished by their labels but also by the type of containment they stand for. While a query node reached by a *rigid edge* (solid line in figure 2.1), which represents direct containment (see definition 5 on page 12), may only be matched by the children of document nodes matching the query node's parent, a *soft edge* (dashed line in figure 2.1) linking two query nodes indicates that the child query node can be matched by any descendant of a document node matching its parent in the query tree. In other words, soft edges represent indirect containment (see definition 6). In terms of the XPath query language [CD99], rigid edges correspond to the `child::` axis (/ in the abbreviated syntax), whereas soft edges are expressed by the `descendant-or-self::` axis (or // for short). Note that like `descendant-or-self::` in XPath, soft tree query edges are reflexive, i.e. the same document node may match both a query node and its child node, provided they are linked by a soft edge.

The textual selection criteria of a tree query, represented by dedicated *textual query nodes* as shown in figure 2.1, specify which keywords a structure hit must contain to qualify also as a *content hit* and enter the final path joining procedure (see below). The query keywords are attached to the incoming edges of textual query nodes, like labels in the case of structural query nodes. Textual query nodes are leaves in the query tree by definition. In terms of the XML Information Set data model (see section 2.2), they are matched by the text nodes below structural document nodes, which in turn match a textual query node's parent. Incoming edges of textual query nodes may be either soft or rigid, just like edges of structural query nodes. A textual query node reached by a rigid edge labelled *"index"* specifies that any document node matching the query node's parent (which is a structural node) is required to contain at least one occurrence of the keyword *"index"*. Such a keyword occurrence in a structurally matching document node is called a *direct keyword occurrence*. If the textual query node's edge is soft, occurrences in descendants of matching document nodes (so-called *indirect keyword occurrences*) are also accepted. Soft edges to textual query nodes translate to the `contains()` function in XPath predicates. For instance, the right path of the query tree shown in figure 2.1 *(b)* could be written `/book/*[contains(., "XML")]`. As a shorthand, indirect containment of keywords is denoted `/book/*[.//"XML"]` in the remainder of this work. Analogously, direct textual containment, as needed e.g. for the right path in figure 2.1 *(a)*, is written `/book//*[./"XML"]` or else `/book//*["XML"]` instead of the somewhat lengthy XPath expression `/book//*[text()[contains(., "XML")]]`. Note that these shorthand notations are not part of the XML Path Language as specified in [CD99].

Textual query nodes may contain multiple query keywords in a conjunction or disjunction. In the former case, all keywords are required to occur within the same matching document node, whereas in the latter case an occurrence of just one of the keywords suffices for the containing document node to qualify as a content hit. By convention, textual query nodes must specify at least one keyword. However, query paths ending in a structural instead of a textual query node are allowed. In this case, no textual selection criteria are given, and document nodes with a matching path are selected regardless of their textual content. (This "don't care" mechanism can be considered the textual equivalent of the structural wildcard $*$.)

Obviously query paths without a textual leaf node, i.e. without query keywords, are less selective than query paths specifying keywords to be retrieved. But a query path's selectivity also depends on which keywords are queried. If a keyword occurs just once in the document collection, there is at most one matching document path to be found, and if this path does not match the structural query criteria, the query is unsatisfiable. Hence rare keywords make a query more selective. Conversely, query keywords which occur most often in the documents being queried hardly ever cause a structurally matching document path to be rejected. In other words, they keep the query rather unselective (although this may be compensated for by restrictive structural criteria). Further examination reveals that query keywords may be selective or unselective with respect to either the number of document nodes they occur in, or the number of different label paths leading to the occurrences. The following two definitions capture this observation:

Definition 7 (NODE SELECTIVITY) A keyword is said to be *node-selective* if there are few document nodes containing an occurrence of this keyword. Conversely, a keyword is *node-unselective* if it occurs in many different document nodes.

Definition 8 (PATH SELECTIVITY) A keyword is said to be *path-selective* if the document nodes where it occurs mostly have identical label paths. Conversely, a keyword is *path-unselective* if there is a great number of different label paths under which an occurrence of the keyword can be found.

But there are other aspects of a query's selectivity besides the frequency of its keywords. Obviously soft edges, whether reaching structural or textual query nodes, represent less selective query criteria than rigid edges. Processing soft-edged query nodes usually involves an exhaustive search in the document tree (or, to anticipate section 2.4.3, in the index tree if one is available). In case of a soft edge to a structural query node, all document subtrees rooted at a node which matches the query node's parent must be searched for document nodes with the right label. A rigid edge instead confines the search to the top level of each such subtree. Similarly, when matching a soft-edged textual query node the whole subtree spanned by any document node which matches the query node's parent has to be scanned for nodes where the query keywords occur. Again, if the textual query node is reached by a rigid edge instead only the children of the subtree's root need to be examined. Hence structural properties of a query tree may also have a considerable influence on how selective the query is, and consequently on how long it takes to process it. In section 5.2.3 of the experimental evaluation, a query classification scheme is presented which strives to formalize query properties such as path and keyword selectivity for later performance comparison.

Finally, a few notes on evaluating query trees as opposed to query paths are in order. If a query consists of more than one path, such as the two sample queries in figure 2.1 on page 13, the individual query paths are processed separately. As described above, this results in a set of matching document nodes for every node in each of the query paths.[5] Two query paths which are part of the same query tree always have at least one node in common. In figure 2.1 *(a)*, e.g., this is the query tree's root, but there might also be a larger path prefix shared by a pair of paths. For any two query paths to be joined, the one among their shared nodes which is lowest in the hierarchy is fixed as their *path join node*. Joining the two query paths, i.e. identifying those of their respective matching document paths which together form a tree, is done by intersecting the two sets of document nodes matching the path join node. Any document node in the intersection is guaranteed not only to match the path join node, but also to belong to two document paths matching either of the two query paths to be joined. Hence there exists a subtree of the document tree, rooted at the node in the intersection, which matches both query paths together. Joining any pair of paths in the query tree this way eventually yields a subtree of the document tree which matches the whole query tree, and thus qualifies as a hit for the entire query. Section 2.4.3 exemplifies this procedure for the DataGuide, an index structure to be presented in the next section.

Throughout this work, query path expressions are typeset in `teletype` instead of sans-serif font to distinguish them from document paths. The slightly modified XPath notation described above is used to represent tree queries. As a path query language, XPath is not very elegant as far as branching is concerned.[6] Multiple query paths all need to be nested in a single predicate attached to the join node. For instance, the expression `/book[./preface/paragraph["index"] and .//*["XML"]]` represents the query tree shown in figure 2.1 *(a)* above, while the tree depicted in *(b)* is written as `/book[./preface/paragraph["index"] and ./*[.//"XML"]]` (mind the shorthand for direct and indirect textual containment, respectively). Alternatively, diagrams similar to figure 2.1 on page 13 may be given, with the same conventions for soft and rigid edges as above (solid lines for rigid edges, dashed lines for soft edges). Query trees are distinguished from document trees and index

[5] Depending on the underlying index structure, this may involve a *path reconstruction* step, which means that given a matching document node, say for a textual leaf in the query path, its ancestors matching the query nodes higher in the path must be determined. The path reconstruction problem is discussed briefly in section 5.4.4.

[6] An alternative syntax for the path query formalism introduced in this chapter is used in appendix B.

trees (to be introduced in the next section) by the shape of their nodes (rounded rectangles, as opposed to circles or ovals and rectangles) and their IDs (prefixed $ instead of & or #).

2.4 DataGuide: an index for Semi-Structured Data

The most influential and well-known index structure for Semi-Structured Data is the *DataGuide* [GW97a]. As the title of this work suggests, the approach to content-aware structure indexing proposed herein is a variation of the DataGuide, like most of the related approaches discussed in chapter 6. This section presents the original DataGuide as a structure index with an additional data structure for content handling, as it has been described in [GW97a] and used e.g. in the LORE system [MAG+97].

2.4.1 Data structures

The DataGuide in its simplest form is a label path index, i.e. it supports path matching, but not keyword matching. Its main data structure is a tree (or graph when indexing graph documents) consisting of all label paths which occur in the indexed document collection. Instead of matching a given query path in the document tree, the index tree is used which is usually much smaller than the former and therefore resides in main memory. An important characteristic of the DataGuide, in contrast to other index structures such as the T-Index [MS99b] or the CIS Index (see sections 4.1.2 and 6.5), is that each label path from the document collection appears exactly once in the index tree. In other words, when two label paths in the document collection share the same prefix (which may well comprise the whole path), the part which is common to both appears only once in the DataGuide, as if the document paths had been merged during index creation. This further simplifies path matching. Figure 2.2 illustrates the collapsing of multiple document paths into a single index path. The document tree shown in *(b)* contains two /book/chapter/section/paragraph paths, one leading to the node &5 and the other to &8. The corresponding DataGuide tree is given in figure 2.2 *(e)*. (Henceforth index nodes are depicted as rectangles, identified by integers with a # prefix.) Note that the two identical label paths have been merged so that there is only a single /book/chapter/section/paragraph path in the index tree, which is therefore usually smaller than the document tree. In the following, we refer to this compact representation of all label paths in a document collection as the *navigational structure* of the index.

The navigational structure alone allows to determine whether a given label path appears in the document collection, but not which document nodes are reached by it. To this end, each index node in figure 2.2 *(e)* needs to reference all document nodes reached by its label path, as shown in *(f)*. For instance, the index node #5 contains references to the two document nodes with the label path /book/chapter/section/paragraph, &5 and &8 (see above). [GW97a] has coined the term *annotations* for this kind of reference. It is important to observe that the annotations do not encode hierarchical relationships between individual document nodes. For instance, it is impossible to tell from the DataGuide's annotations shown in figure 2.2 *(f)* whether the document node &5 is a child of &4 or &7, which are both referenced by the parent of &5's referencing index node, #4. For this *path reconstruction*, which is often required when representing query results, secondary indices or other techniques are needed (see section 5.4.4 for a short discussion of the path reconstruction problem).

Also note that while the annotated navigational structure, i.e. the index tree with document node references, is a useful conceptual representation of the DataGuide, it is usually not stored this way on disk. (Section 2.4.4 deals with this issue in more detail.) Rather the annotations are kept in a table mapping a given index node to the document nodes it references, just like the one shown in figure 2.2 *(c)*. Formally, this *annotation table* implements a mapping $dg_a : i \mapsto D^d_{k,i}$ where i is a an index node, k a keyword, and $D^d_{k,i}$ the set of document nodes with the same label path as i where k occurs.[7]

[7] The superscript d in $D^d_{k,i}$, which seems superfluous here, is used in section 3.3.1 to distinguish the DataGuide's

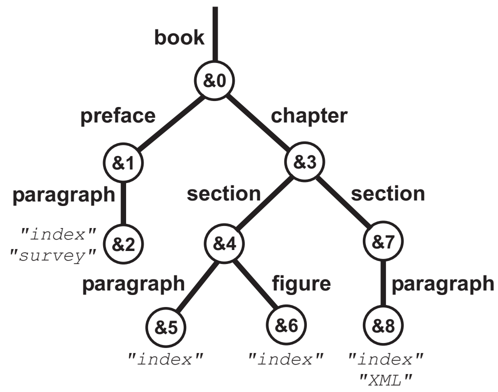
```
<?xml version="1.0" ?>
<book id="_0">

  <preface id="_1">
    <paragraph id="_2">
      index survey
    </paragraph>
  </preface>

  <chapter id="_3">
    <section id="_4">
      <paragraph id="_5">
        index
      </paragraph>
      <figure id="_6">
        index
      </figure>
    </section>
    <section id="_7">
      <paragraph id="_8">
        XML index
      </paragraph>
    </section>
  </chapter>

</book>
```

(a) XML document



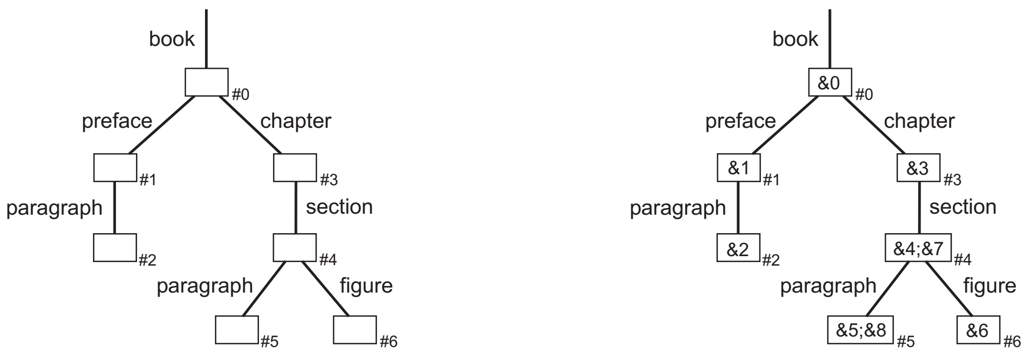
(b) tree representation of *(a)*

#0	#1	#2	#3	#4	#5	#6
&0	&1	&2	&3	&4;&7	&5;&8	&6

(c) annotation table

"index"	*"survey"*	*"XML"*
&2;&5;&6;&8	&2	&8

(d) content table



(e) navigational structure *(f)* annotated navigational structure

Figure 2.2: Data structures of the DataGuide

As mentioned above, the DataGuide in its simplest form ignores the textual content of the documents being indexed. A secondary *content index* serves to evaluate structure queries with keywords, as the ones discussed in section 2.3. A content index is simply a mapping from keywords to document nodes, which allows to retrieve all nodes in the document tree where a given keyword occurs. For instance, a content index built over the document collection in figure 2.2 *(b)* should map the keyword *"survey"* to the document node `&2`, and the keyword *"index"* to the set of document nodes containing `&2`, `&5`, `&6`, and `&8`. Analogously to the index node annotations discussed in the previous paragraph, the content index is held in a *content table* as the one shown in figure 2.2 *(d)*. Formally, it realizes a mapping $dg_c : k \mapsto D^d_{k,i}$ where k is a keyword, i an index node, and once again $D^d_{k,i}$ the set of document nodes reached by i's label path which also contain an occurrence of k. The DataGuide shown in figure 2.2 *(f)*, used together with such a content table in the retrieval algorithm described in section 2.4.3, can then process path queries like e.g. `/book/preface/paragraph["index"]` or `/book//` $*$ `["XML"]`, which may also combined to yield tree queries as the ones depicted in figure 2.1 above.

2.4.2 Indexing algorithm

[GW97a] describes a straightforward indexing algorithm for the DataGuide, designed to handle arbitrary graph-shaped document collections. Since this work only considers document trees, as mentioned in section 1.1, the procedure for creating the DataGuide's navigational structure can be further simplified. On the other hand, indexing keywords simultaneously with label paths requires minor modifications. The resulting indexing algorithm for tree document collections, being conceptually very close to the original one, is only sketched briefly here.

```
// recursively traverses the document tree, building a DataGuide simultaneously
// initially called for the document tree's root and the newly created DataGuide root
procedure createDG (DocNode d, IndexNode i)
  for all children d_c of d do

    // update content table
    for all keywords k in the set of d_c's keywords do
      add d_c to dg_c(k)
    end for

    // update annotation table
    l := d_c's label
    if i has a child i_l reached by an edge labelled l then
      i_c := i_l
    else
      i_c := a new index node, reached from i by an edge labelled l
    end if
    add d_c to dg_a(i_c)

    // continue recursion
    call createDG (d_c, i_c)

  end for
end procedure
```

Algorithm 2.1: DataGuide creation

basic annotation mapping from an optimized one.

The algorithm in listing 2.1 shows how to build a tree-shaped DataGuide in a recursive manner. At indexing time, the document tree is traversed in a depth-first fashion. The DataGuide to be created is constructed and navigated in parallel, node by node. In the beginning, when following the first edge leaving the document tree's root node, a newly created node is added to the index tree and linked to its root node by an edge with the same label. Then the procedure continues analogously with the document node reached by the edge just processed, and with the newly created index node. Further index nodes are only added when following a label path in the document collection which does not yet exist in the DataGuide. Otherwise the index node with the right label path is used, which avoids duplicate paths in the index tree. Every document node being indexed during the depth-first traversal of the document tree is added to the corresponding index node's annotation set in the annotation table. Besides, for each keyword occurring in the document node a new entry in the content table is created, if necessary. The document node is then added to the node lists of all these keywords. (Recall from the previous subsection and figure 2.2 *(d)* that the content table contains for each keyword the list of document nodes where this keyword occurs.) When the whole document tree has been processed this way, the DataGuide is complete, and possible postprocessing procedures of the navigational structure (such as the index node ID computation discussed in section 3.3.3) may take place.

2.4.3 Retrieval algorithm

The DataGuide's retrieval procedure can be divided into the following four *retrieval phases*:

phase 1: *navigation*

phase 2: *annotation fetching* and *keyword occurrence fetching*

phase 3: *content/structure join* of keyword occurrences and annotations

phase 4: *path join* in case of a tree query

As mentioned in section 2.3, tree queries are split into individual query paths, which are processed separately and finally joined to yield hits matching the entire query tree. In the first retrieval phase, the labels of a given query path are matched sequentially to the DataGuide's navigational structure, until either a structural mismatch occurs or an index node with a matching label path is found. When the given query path contains soft edges or edges with the wildcard label $*$, there may be more than one such index node. If the index tree does not contain paths matching the query path, the query is unsatisfiable, and the retrieval procedure stops. Otherwise one needs to find out which of the document nodes referenced by a matching index node from phase 1 contain the keywords attached to the given query path. To this end, the annotations of every matching index node are fetched from the database. Additionally, the occurrences of all query keywords are retrieved for content matching. This double database access makes up the second retrieval phase. Phase 3 brings together the results from structure and content matching: the two fetched sets of document nodes are intersected to identify for every given query path the set of document nodes with the right label path and keywords. The retrieved document nodes match the leaf node in the corresponding query path. When multiple paths form a query tree, however, these leaf hits need to be propagated upwards along their respective query paths, and combined whenever a path join node is reached (see section 2.3). Note that even though all query paths are evaluated separately, phases 1 to 3 on the one hand and phase 4 on the other hand may be interleaved. In this way, newly processed query paths are immediatly joined with the ones already finished, which can save repeated navigation of common path prefixes.

Listing 2.2 contains the DataGuide's retrieval algorithm for the most simple query paths, which do not contain soft edges or unspecified labels. The individual retrieval phases are marked in the code. Note that retrieval phase 4, where multiple query paths are joined, has been omitted in this procedure. It depends partly on the kind of result data structure to be constructed, and hence on the particular implementation of the underlying retrieval system (see chapter 4).

```
// recursively matches a label path with query keywords against the index tree, starting from a given node
// returns the set of document nodes reached by the given label path where the given keywords occur
procedure processQueryPathDG (LabelPath p, Set K, IndexNode i)

  // retrieval phase 1: navigation
  if p has more edges to be matched then
    l := the label of p's first edge
    if i has a child i_l reached by an edge labelled l then
      p' := p with the first edge chopped off
      return processQueryPathDG (p', K, i_l)
    else
      return ∅
    end if
  end if

  // retrieval phase 2: annotation and keyword occurrence fetching
  C := dg_c(K)
  A := dg_a(i)

  // retrieval phase 3: content/structure join
  return C ∩ A

end procedure
```

Algorithm 2.2: Retrieval with the DataGuide (simplified)

As an example, reconsider the DataGuide from figure 2.2 and the query tree depicted in figure 2.1 *(a)* on page 13. It consists of two paths which are processed separately in the retrieval phases 1 to 3, before being joined in phase 4, as follows. The first label path in the query tree, `/book/preface/paragraph`, is searched in the navigational structure from figure 2.2 *(e)*. There is a single index node, #2, which passes the navigation phase, its label path matching the one of the query path. In phase 2, its annotations are fetched from the annotation table shown in figure 2.2 *(c)*, $dg_a(\#2) = \{\&2\}$. Likewise, the occurrences of the query keyword *"index"* are looked up in the content table from figure 2.2 *(d)*, which yields $dg_c(\text{"index"}) = \{\&2; \&5; \&6; \&8\}$. In retrieval phase 3, $dg_a(\#2) \cap dg_c(\text{"index"}) = \{\&2\}$ is identified as the set of document nodes matching the query path `/book/preface/paragraph["index"]` or, to be more precise, this path's lowest structural node, which has the ID `$2` in figure 2.1. The second path in the query tree, `/book//*["XML"]`, is more complex due to its soft structural edge linking the query node `$0` to its child node `$4`. Starting from its root node #0, the whole index tree is searched for nodes whose label path matches the query path `/book//*`. The set of index nodes to be examined in retrieval phase 2 comprises all index nodes (including #0 itself because soft structural edges are reflexive, as specified in section 2.3). Yet the intersection operation in phase 3 produces empty hit sets for most of them, their annotation sets being disjoint with the occurrence set for the query keyword *"XML"*, $dg_c(\text{"XML"})$. Only #5's annotation set shares a single document node with $dg_c(\text{"XML"})$, namely &8 (which is indeed the only node in the document collection where the keyword *"XML"* occurs, as can be verified in figure 2.2 *(b)*).

As described in section 2.3, retrieval phase 4 involves a path reconstruction step which serves to identify candidate document nodes matching the path join nodes in the query tree, based on the hits retrieved for the individual query paths. Those candidates which are not confirmed by all query paths leaving a given path join node are discarded. In the case of the two query paths from the example, this means that for the whole query to be satisfied, an ancestor node common to both &2 and &8 (the hits retrieved in the previous retrieval phase) needs to be found at the document tree's root level, where it is guaranteed to match the join node `$0` linking the two query paths. Obviously there is one such document node, &0. Hence the two reconstructed paths can be combined to form a subtree of the document collection in figure 2.2 *(b)* matching the tree query from figure 2.1 *(a)*. The result is shown as a Complete Answer Aggregate (see section 4.1.3) in figure 3.7 on page 40.

The above example illustrates that selective query keywords may act as a filter to unselective query paths. During retrieval phase 2, annotation fetching for the second query path involves many database accesses, simply because there are many matching paths in the index tree. When joining structure and content matches in phase 3, however, most of these fetching operations are proved futile. Obviously the strict separation of content and structure matching during the first two retrieval phases is inconvenient in such cases. Note that a reversed matching order – first keyword fetching, followed by navigation and annotation fetching – has no effect, unless keyword fetching fails altogether (in which case navigation is useless, and the query can be rejected as unsatisfiable right away). Moreover, it results in similar deficiencies for queries with selective paths, but unselective keywords. In other words, the DataGuide faces an inherent defect because it keeps structural and textual selection criteria apart during the first two retrieval phases. The goal of this work is to develop a *Content-Aware DataGuide* combining structure and content matching from the very beginning of the matching process. Chapter 3 presents several approaches to this task, which are evaluated empirically in chapter 5.

2.4.4 Storage management

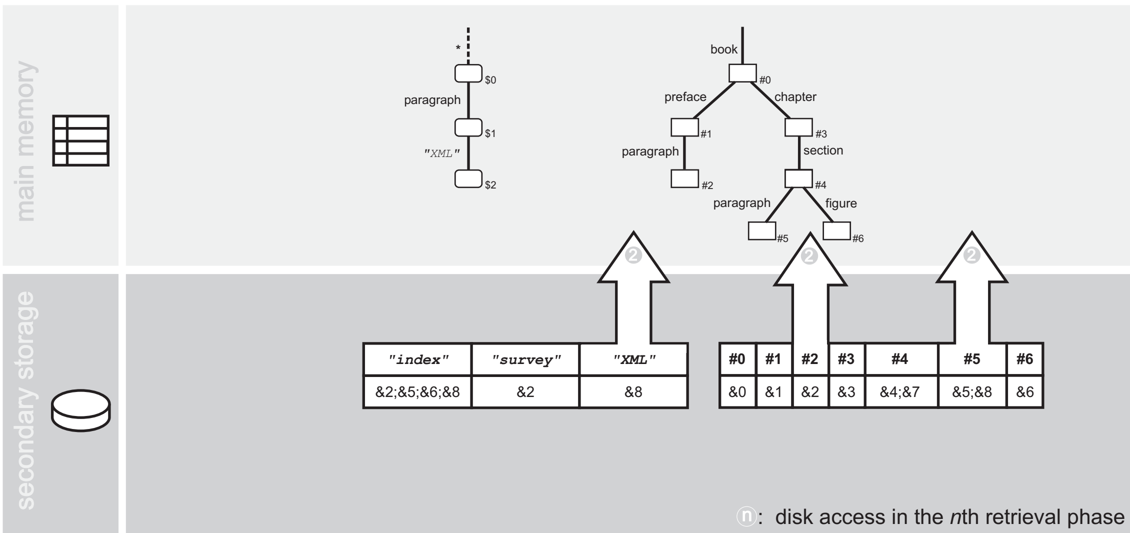


Figure 2.3: Storage management of the DataGuide

Figure 2.3 illustrates the different data structures involved in query processing and their distribution among different storage devices. Secondary storage is managed by a relational database back-end. The query tree (symbolized by a single query path in the figure) is kept in main memory during the evaluation. Likewise, the DataGuide's navigational structure can be expected to fit in main memory—except perhaps for huge document collections following an extremely diverse, probably recursive schema. However, this can be neglected for most real-world scenarios.[8] By contrast, the DataGuide's content and annotation tables are usually too large to be held in main

[8] The experimental evaluation documented in chapter 5 includes a recursive document collection of more than 500 MB. The resulting DataGuide, which has nearly 2,500 nodes, only requires about 5 MB of main memory.

memory. Depending on the indexed document collection, the content table often contains multiple references to almost every document node. In a DataGuide's annotation table all document nodes are guaranteed to be referenced, although only once in the case of a tree-shaped document collection. Hence the two tables need to be stored on disk, which increases the cost of occurrence fetching considerably. As shown in the figure, they are accessed only in retrieval phase 2, but possibly more than once (depending on both the index implementation and the query being evaluated).

2.5 Text indexing with signatures

Signatures are a common data structure from Information Retrieval [Fal85, DML98], designed for concise representation and fast processing of heuristic information. They serve as a basis for the Signature CADG presented in section 3.2.3.2. At first sight, signatures are simply bit strings of uniform length, and as such sufficiently general to encode any kind of information, e.g. about the keywords occurring in a document node. Their heuristic nature results from the fact that usually there is not enough space available to represent the exact data as a bit string. Hence every signature only holds part of the original information. Further approximation is achieved by combining the information stored in multiple signatures to encode larger ensembles of data, such as all keywords occurring in an entire subtree of the document tree, rather than a single node. To this end, signatures are disjoined, i.e. combined by applying the disjunction operator $\sqcup$ to all bits separately. The result is a new signature with the same number of bits as the ones which have been combined, but a possibly different value. There is a `1` at every position in the resulting bit string which was set in one of the original signatures, even if all other signatures had a `0` at this position. Metaphorically, one can say that every bit set in any of the signatures being disjoined "leaves its trace" in the newly created signature. Figure 2.4 exemplifies this procedure for two pairs of keywords, illustrating how signature disjunction can cause a loss of information. Each keyword is assumed to have been assigned a (preferrably unique) signature before, as shown in the figure. Then the signatures of either pair of keywords are combined in a bitwise disjunction ($\sqcup$) to produce a new signature, which represents two keywords simultaneously.

"index"	`01100000`	$\neq$	`01000100`	*"midi"*
"XML"	`00000110`	$\neq$	`00100010`	*"min"*
"index" $\sqcup$ *"XML"*	`01100110`	=	`01100110`	*"midi"* $\sqcup$ *"min"*

Figure 2.4: Ambiguous signatures

Figure 2.4 shows how every keyword participating in the disjunction "leaves its trace" at those positions where it has a `1` in its signature. However, when multiple keyword signatures have a bit set at the same position, a document's signature does not reveal which ones left a trace and which ones did not. For instance, one cannot tell from the signature $\boxed{\texttt{01100110}}$ whether the word "index" occurs in the corresponding document, since a couple of other keywords together produce the same signature, as shown in the figure above. Obviously such ambiguities arise even when using unique keyword signatures. Of course signatures could in theory represent the exact keyword information. A naive way to achieve this is to use bit strings with a dedicated bit for each keyword occurring in the document collection. Yet this is obviously unfeasible for realistic document collections. When exact content representation is out of scope, there exists a trade-off between the number of keywords to be represented, the signature length, and the precision of the information encoded by the signature. Fixing one of the three factors leaves two antagonists. For instance, given a fixed number of keywords to be represented, the signature's precision decreases with the number of bits per signature. Likewise for signatures of a given length, precision declines as more keywords are represented.

In the following, the bitwise disjunction is denoted by $\sqcup$, as above, and the bitwise conjunction by $\sqcap$. $\neg$ symbolizes the bitwise inversion. Additionally a function $\sqsubset$, called *bitwise implication*, is defined as $s_0 \sqsubset s_1 := \neg s_0 \sqcup s_1$, for any two signatures s_0 and s_1. In other words, the value of $s_0 \sqsubset s_1$ is a signature in which only those bits are set to `1` which either are also set to `1` in s_1 or to `0` in s_0. The bitwise implication can be used to define a Boolean function whose value is `true` only when the signature resulting from the bitwise disjunction has all bits set to `1`, and `false` otherwise. For the sake of simplicity, this boolean function is sometimes written $\sqsubset$ in the remainder of this work, just like the bitwise disjunction.

Chapter 3

Content-Aware DataGuide (CADG)

3.1 Objectives

Section 2.4 has introduced the DataGuide, mentioning its advantages over other, especially non-navigational index structures, but also its shortcomings. A major drawback of the DataGuide, e.g. compared to the IndexFabric (see section 6.2), Signature File Hierarchy (see section 6.3), or BUS index (see section 6.4), is its strict separation of structure matching and content matching.[9] Depending on the underlying data, the textual part of a query can be much more selective than the structural part, i.e. the label paths leading to keyword occurrences. Hence there is a considerable optimization potential left aside by the DataGuide.[10] The Content-Aware DataGuide (CADG) strives to address this weak-spot by integrating content matching with both phases of structure matching, namely navigation and annotation fetching, thus making the content/structure join (phase 3) superfluous. In short, one could say that the CADG is a DataGuide, enhanced with a materialized content/structure join and a keyword-specific path matching procedure. These enhancements require substantial reorganization of both the data structures of the index and its retrieval algorithms. Before examining the proposed solution in detail, this section summarizes the two main goals of the Content-Aware DataGuide. They will be reviewed in chapter 5 when evaluating the experimental results.

3.1.1 Content-aware index navigation

The first objective is to minimize the computational effort caused by index navigation during query evaluation. The idea is to *prune*, i.e. omit, index tree paths referencing parts of the document collection where the query keywords do not occur. As mentioned in section 2.4, the retrieval algorithm of the ordinary DataGuide, ignoring content information during path matching, may waste time following paths in the index tree which are bound to fail in the subsequent content matching step. Much worse, fetching occurrences for a given index node involves an expensive database access. Hence each index path which is not pruned although it does not lead to relevant keyword occurrences causes needless I/O activity, apart from the time it takes to match the path to the query. In other words, taking the query keywords into account during path matching may help to avoid both needless in-memory and secondary storage operations, provided the keywords are sufficiently path-selective to bring pruning into effect. Anticipating section 5, where different indexing and querying scenarios are explored, one can expect that content-aware navigation enhances the evaluation performance mainly for queries with highly path-selective keywords.

[9] As remarked in section 2.4.1, the original DataGuide ignores content indexing altogether, serving as a browsable path index only.

[10] For an overview of earlier proposals to address this issue, see chapter 6.

As discussed in section 3.2.3 below, there are several ways to integrate content information with path matching. The two approaches proposed in this work differ in their representation of content, which crucially affects pruning precision and evaluation efficiency. While content-awareness based on index node IDs (see section 3.2.3.1) allows for exact pruning, signature-based content-aware navigation (see section 3.2.3.2), relying on efficient but heuristic bit operations, cannot guarantee to prune all irrelevant index paths.

3.1.2 Content-aware annotation fetching

The second objective is to avoid the content/structure join which brings together the results from retrieval phases 1 and 2. This can be accomplished by fetching only those structure hits from the database which also contain the given query keywords. By contrast, the DataGuide's retrieval algorithm, as discussed in section 2.4.3, collects all structure hits first, regardless of their content, and then joins them with all occurrences of the given query keywords in a second step. This work proposes two ways to make the annotation fetching step content-aware, as described in section 3.2 below. In both cases joining structure and content matching hits no longer requires an intersection of possibly huge sets of document nodes at query time, but only a simple selection from a content-aware annotation table precomputed at indexing time. In other words, the content/structure join (retrieval phase 3) is lifted from the level of document nodes to the index node level. This feature should be useful when querying hardly node-selective keywords, whose entries in the content table contain many document node IDs. Furthermore, since structure and content are kept in the same table, content-aware annotation fetching requires only a single disk access, whereas in the case of the ordinary DataGuide the content and annotation tables are solicited separately.

3.2 Conceptual view

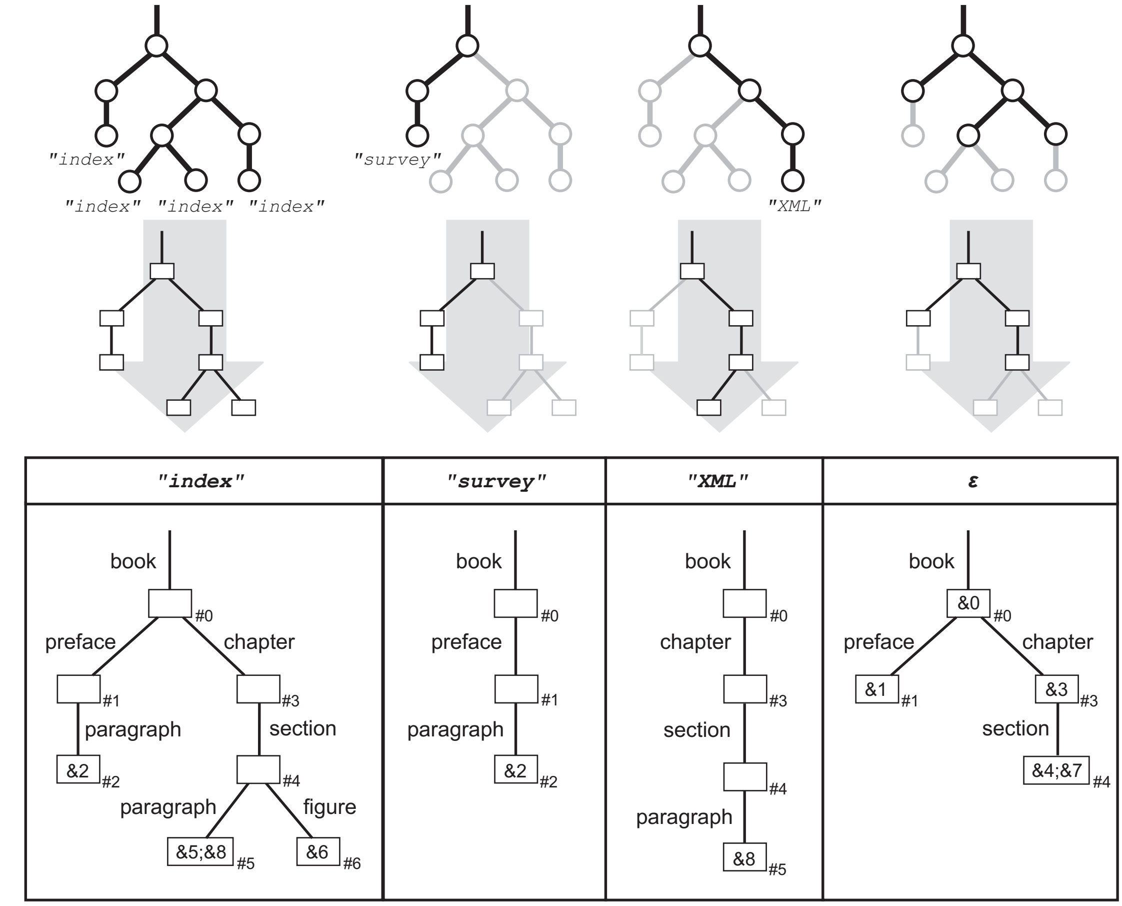
This section presents the Content-Aware DataGuide from a conceptual point of view. Focussing on its essential data structures and the retrieval algorithm, the following description leaves aside implementation details, which are covered in chapter 4. Suffice it to say that like in the case of the DataGuide, a relational database back-end is used to store parts of the index. The indexing algorithm of the CADG, as a variation of the ordinary DataGuide, is similar to the one sketched in section 2.4.2 and therefore not repeated here.

A first naive approach to content-aware structure indexing serves as an intuitive starting point, but reveals defects which make it both time and space-inefficient. It serves merely to motivate the following work and is not pursued any further. In a second step, a more sophisticated solution is proposed on a high level of abstraction. Finally, two alternative index structures based on this concept are developed, namely the *Signature CADG* and the *ID-Comparison CADG*. They differ mainly in the way content-aware navigation is realized. Both are compared to each other and to the ordinary DataGuide in terms of their respective data structures, retrieval algorithm, and storage behaviour.

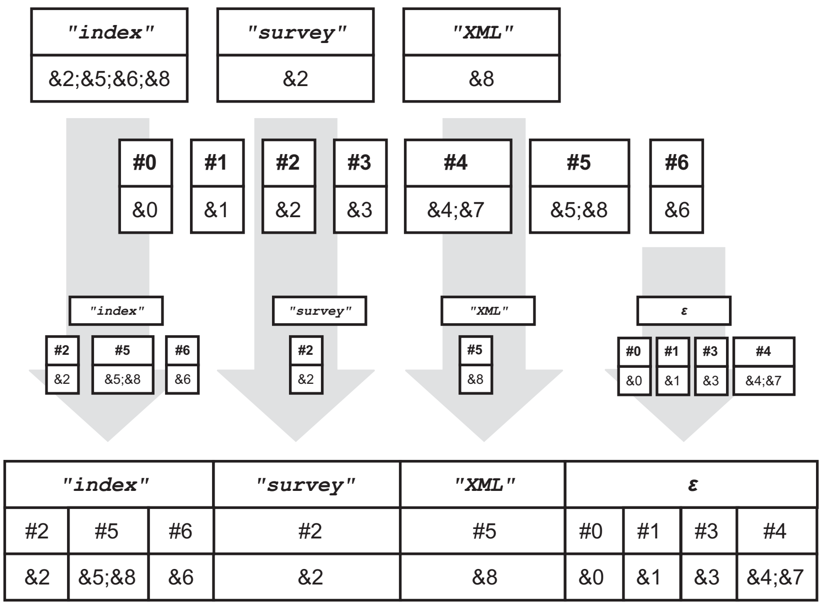
3.2.1 Starting point: the content-centric approach

3.2.1.1 Data structures

As described in section 2.4, the ordinary DataGuide's retrieval algorithm goes through four phases, namely first index navigation, second annotation and keyword occurrence fetching, third content/structure join, and fourth path join. The principle idea of the Content-Aware DataGuide is to precompute the content/structure join such that during the first two retrieval phases, structure and content information is processed simultaneously. Clearly the data structures used by the index must be adapted to support this integration of document structure and content. On the one hand, the navigational structure needs to be enriched with keyword information allowing for content-aware navigation, as claimed in section 3.1.1 above. On the other hand, recalling the



(a) content-centric navigational structure



(b) content-centric content/annotation table

Figure 3.1: Data structures of the content-centric approach to content-aware indexing

second objective described in section 3.1.2, the content and annotation tables must be combined to support content-aware annotation fetching.

Figure 3.1 illustrates one way to carry out these two adaptions of the ordinary DataGuide, again for the document collection shown in figure 2.2 *(b)*. The idea, in short, is to build a separate DataGuide for each keyword occurring in the document collection. This approach is *content-centric* in the sense that the original DataGuide's navigational structure and tables are reorganized at indexing time to enable fast content matching, treating structure as a subordinate criterion. The navigational structure shown in figure 2.2 *(f)* falls apart into a forest of keyword-specific subtrees each covering only those document nodes which contain the respective keyword *(a)*. As suggested by the figure, one can think of such a sub-DataGuide as indexing a view of the document collection which is induced by a single keyword, and which excludes the paths leading to document nodes where this keyword does not occur. One subindex might consist of a single path, e.g. /book/preface/paragraph for the keyword *"survey"* in *(a)*, if the keyword defining the corresponding view only occurs in document nodes reachable via this label path. Another subindex, say for the keyword *"index"*, could well comprise all available document paths, being isomorphic to the ordinary DataGuide for this collection, provided each label path reaches a document node where the word *"index"* occurs. The subindex in the rightmost column, labelled ε (the empty word), contains references to those document nodes which do not contain keywords. In the example, these are the five inner nodes of the document tree, &0, &1, &3, &4, and &7, which are indexed by the four inner nodes of the index tree, #0, #1, #3, and #4. It is needed for pure structure queries.

Figure *(b)* shows how the DataGuide's content and annotation tables can be combined to yield a single content/annotation table. Similar to the splitting of the navigational structure, the content table from figure 2.2 *(d)* is divided into keyword-specific subtables. In this case there are three such subtables, one for each keyword. To integrate them with the annotation table's information about index node/document node pairings (see figure 2.2 *(c)*), each keyword's list of document node IDs is grouped by the referencing index nodes. Again all references to document nodes without keywords are subsumed under the label ε.[11] The resulting content/annotation table shown at the bottom of figure 3.1 *(b)* represents a mapping $cadg_{cc} : (k, i) \mapsto D^d_{k,i}$ where k is a keyword, i is an index node ID, and $D^d_{k,i}$ is the set of document nodes where k occurs and which are referenced by i. For instance, the document node IDs &2, &5, &6, and &8, which are kept together in the *"index"* column of the DataGuide's content table (topmost table in figure 3.1 *(b)*, or figure 2.2 *(c)*), are distributed over three columns in the combined content/annotation table (figure 3.1 *(b)* bottom) while still being subsumed under the keyword *"index"*: &2 is in the subcolumn of #1, which is its referencing index node, &5 and &6 together belong to #5, and &6 is in the #6 subcolumn.

Note that the annotations of each subindex in figure 3.1 *(a)* correspond to exactly one of the keyword columns in the content/annotation table *(b)*. For instance, the annotations of the *"index"* subindex, &2, &5, &6, and &8, appear in the content/annotation table's leftmost keyword column, *"index"*. Intuitively, the subindex for a keyword k_0 has its own subtable covering just those index nodes which reference document nodes containing an occurrence of k_0. Just as the subindex for k_0 is actually a DataGuide indexing only those document nodes where k_0 occurs, the k_0 column in the content/annotation table is actually this DataGuide's annotation table. (Since the sub-DataGuide ignores all other keywords, there is no need for a content table any more.) The content/annotation table in figure 3.1 *(b)* can therefore be described as a content table containing multiple annotation tables. Formally, this nesting becomes obvious when the mapping provided by the content/annotation table, $cadg_{cc} : (k, i) \mapsto D^d_{k,i}$, is represented in its curried form, $k \mapsto (i \mapsto D^d_{k,i})$. This second-order function maps a keyword k to the first-order function $i \mapsto D^d_{k,i}$ which section 2.4.1 introduced as the functional equivalent dg_a of the DataGuide's annotation table.

[11] This procedure corresponds to computing the Cartesian product of the set K of keywords with the set I of index nodes IDs, with three modifications. First, those elements $(k, i) \in K \times I$ with $dg_c(k) \cap dg_a(i) = \emptyset$ are discarded. Second, each of the remaining pairs (k, i) is extended to a triple $(k, i, dg_c(k) \cap dg_a(i))$. Third, for each i not represented in the set resulting from the previous steps, $(\varepsilon, i, dg_a(i))$ is added.

3.2.1.2 Retrieval algorithm

The retrieval process of the above index structure closely resembles the one of the ordinary DataGuide. To bring content-awareness into effect, the keyword selection is integrated with both phase 1 and phase 2. First of all, for each query keyword the corresponding subindex is loaded. Then path matching takes place in each of them as with the ordinary DataGuide, i.e. each query path is matched against the subindex which is appropriate for the path's keyword.[12] This yields a set of index nodes whose occurrences need to be fetched. Note, however, that in contrast to the ordinary DataGuide these index nodes are guaranteed to reference document nodes where the respective keywords occur. The reason is that each query path is matched against a subindex which only contains keyword-relevant nodes, such that false hits matching only structure, but not content cannot occur. In other words, the index structure supports content-aware navigation. In phase 2, for each of the matching index nodes from phase 1 those occurrences are fetched from the content/annotation table which contain the respective query keywords. As described for the previous phase, the existence of at least one such occurrence for each index node is guaranteed. Contrary to the ordinary DataGuide, both the index node and the keyword in question serve as selection criterion during occurrence fetching. This ensure the content-awareness of phase 2, making a subsequent content/structure join (phase 3) unnecessary.

As an example, reconsider the index structure shown in figure 3.1. To process the query `//*/paragraph["XML"]`, first the subindex for the keyword *"XML"* is fetched. It consists of a single path, /book/chapter/section/paragraph. Hence matching the query path `//*/paragraph` does not entail much navigation in this case. Following the index path in top-down direction like in an ordinary DataGuide, one reaches the leaf node #5, which is the only hit retrieved in phase 1. The content/annotation table's *"XML"* column, as shown in figure 3.1 *(b)*, maps index node #5 to document node &8. Thus at the end of phase 2 the only document node matching the given query, &8, has been identified without an explicit content/structure join. In case of a tree query, a path join would be performed just like for the ordinary DataGuide.

3.2.1.3 Storage management

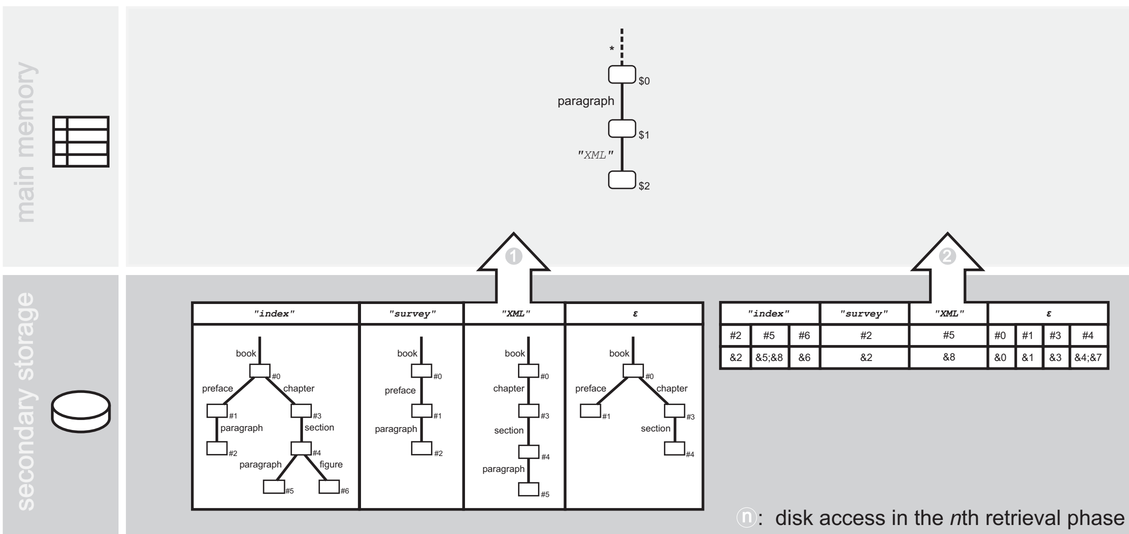


Figure 3.2: Storage management of the content-centric approach to content-aware indexing

[12] Again, we disregard multiple keywords per query path for the sake of simplicity.

As figure 3.2 illustrates, all keyword-specific subindices are supposed to be stored on secondary storage devices. Of course there is a caching effect when some of them have already been loaded into main memory for a previous query. However, this may not be the case for those subindices which are relevant for a given set of query keywords. Therefore the relevant subindices need to be loaded at the beginning of evaluation phase 1 in general. Compare this setting to the one shown in figure 2.3 on page 21 for the DataGuide, whose navigational structure is permanently held in main memory. The content/annotation table, being kept on disk as well, is accessed during phase 2, when the occurrences of all query keywords reachable via the matching label paths are fetched.

3.2.1.4 Discussion

The content-centric approach to content-awareness is based on keyword-based pruning at indexing time. As a consequence, index nodes which are irrelevant for a given query keyword are never encountered during path matching. The DataGuide's matching procedure (see section 2.4.3), by contrast, visits all index nodes during navigation, possibly causing needless disk operations during occurrence fetching. The fetching process itself is enhanced by integrating index node and keyword selection within a single database query, which again reduces the number of disk accesses and avoids intersecting possibly huge sets of document node IDs. However, this combination of structure and content means that the content/annotation table is accessed for all kinds of query. In the case of pure structure or pure content queries, this might take longer than querying either the content table or the annotation table of the ordinary DataGuide: the reason is that in contrast to the one-dimensional keys of the latter two, the content/annotation table has two-dimensional keys and thus requires a more complex database index. However, this effect has not been observed in the experimental evaluation (see chapter 5).

Although the content-centric approach has been shown to meet the requirements from section 3.1.1 and 3.1.2, it has three major drawbacks. First, a subindex whose keyword is not path-selective at all becomes a full-sized DataGuide, as shown in figure 3.1 *(a)* for the keyword *"index"*. In the worst case, the overall size of the corresponding navigational structure is $|K| + 1$ times the size of the corresponding DataGuide, where K is the set of distinct keywords in the document collection[13]. Yet all subindices are stored on disk, such that this redundancy is not a knock-out criterion. What is worse, however, before evaluating a query the navigational structures of the appropriate subindices – one for each query keyword – need to be fetched from the database, since they cannot all be supposed to fit into main memory. This entails additional disk operations which slow down the evaluation process considerably. Depending on how the subindices are stored, more or less expensive parsing and object creation steps are necessary, too. Finally, to evaluate queries with keyword con- or disjunctions, the same query path must be matched against multiple index trees, and the resulting sets of matching index nodes need to be intersected or united before fetching the corresponding occurrences.

The disadvantages of the content-centric approach result mainly from the fact that the index structure is commited to fast keyword retrieval, at the cost of a time and space-efficient organization of the navigational structure. First, maintaining separate DataGuides for all keywords almost inevitably causes redundancy, as the example in figure 3.1 illustrates: five out of six index nodes in the original DataGuide's navigational structure are duplicated in the course of the content-centric adaptations. Not only does this increase the storage overhead caused by the index, but it also contributes to a performance deterioration at query time—the more space each subindex occupies in main memory, the fainter are the chances to have the right one already loaded for a given query keyword. Second, the content-centric approach forces the ordinary DataGuide's navigational structure into a table structure, despite its inherently non-relational nature, in order to access it piecewise by query keyword. It is true that if the sum of all relevant subindices for a given query is smaller than the whole navigational structure, this may save space in main memory at query time, because only those parts of the tree are loaded which need to be navigated anyway. For instance, in case of a fairly path-selective query keyword like *"survey"* in figure 3.1 above, the

[13] As explained before, there is one keyword-specific subindex for each member of K, plus another DataGuide to answer pure structure queries.

corresponding subindex occupies far less space than the content-unaware navigational structure of the DataGuide. However, this is not true for all keywords—consider e.g. the subindex for *"index"*. Furthermore, keeping the whole navigational structure in main memory permanently minimizes the number of expensive disk operations at query time. The content/annotation table, by contrast, is best kept on a secondary storage device. Thanks to its table structure, one can take advantage from highly sophisticated relational database techniques to optimize occurrence fetching.

To summarize, the index structure presented in this subsection, while addressing the main weak-spots of the DataGuide, is not appropriate for real-world query scenarios featuring a wide range of query keywords. Although highly redundant, it does not support efficient query evaluation, keeping virtually all its data structures in secondary storage. The next subsection shows how to overcome these defects while supporting content-aware navigation and occurrence fetching, as claimed in section 3.1.

3.2.2 Refinement: the structure-centric approach

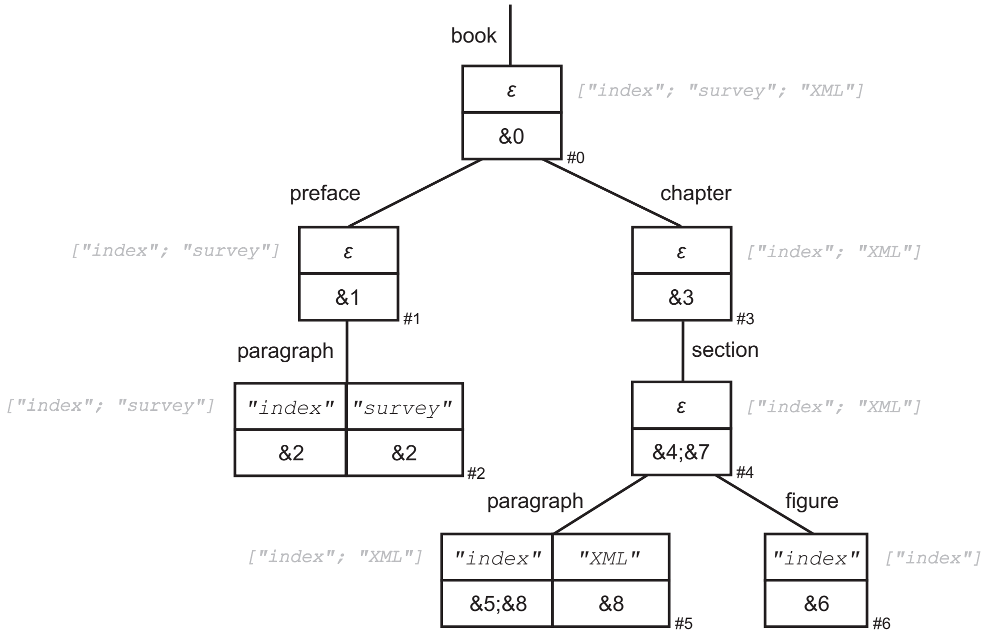
3.2.2.1 Data structures

The preceding discussion explained why for efficient query evaluation, it is advantageous to preserve the navigational structure of the index in its integrity, rather than to split it up into keyword-specific subindices. As argued in section 2.4.4 for the ordinary DataGuide, the entire index graph can be expected to fit main memory unless the underlying document collection's schema is extraordinarily diverse. The approach presented in this subsection makes use of this feature to improve query performance, in this sense being more close to the DataGuide than the content-centric index structure examined in the previous subsection. In fact, the two approaches to content-awareness organize their data structures in a complementary manner, as a comparison of the figures on page 27 and page 32 reveals. While the navigational structure in figure 3.1 *(a)* is partitioned by keyword to support fast content matching, the *structure-centric* approach in figure 3.3 *(a)* favours structure matching by keeping the whole navigational structure together. By contrast, the corresponding content/annotation table in figure 3.3 *(b)* is divided into index-node specific subtables (which have been attached to their respective index nodes in figure 3.3 *(a)*), instead of keyword-specific ones like in figure 3.1 *(b)*. Each of these index-node specific subtables maps a given keyword to the set of document nodes where this keyword occurs, and which are referenced by the index node associated with the subtable.

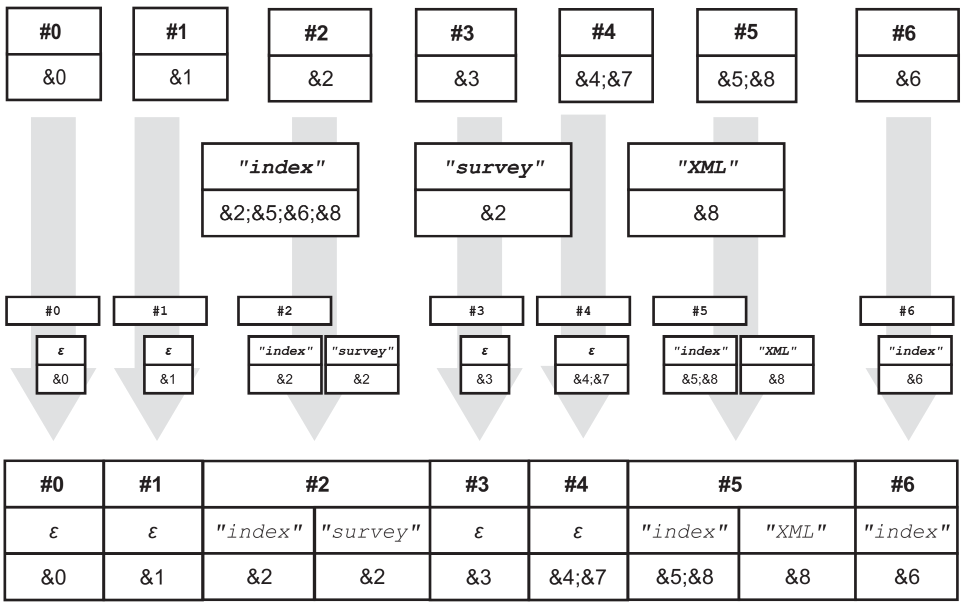
As a whole, the structure-centric content/annotation table in figure 3.3 *(b)* represents the same mapping as the content-centric one (see figure 3.1 *(b)*), namely $cadg_{sc} : (i, k) \mapsto D^d_{k,i}$ where i is an index node ID, k is a keyword, and $D^d_{k,i}$ is the set of document nodes where k occurs and which are referenced by i (see section 3.2.1.1). Compared to $cadg_{cc}$, only the order of the input tuple's components has been reversed. In its curried form $i \mapsto (k \mapsto D^d_{k,i})$, however, $cadg_{sc}$ reveals that the content/annotation table is in fact an annotation table mapping index nodes to their respective content table fragments. (Recall from section 2.4.1 that $dg_c : k \mapsto D^d_{k,i}$ is the mapping represented by the DataGuide's content table.) In other words, the subtable associated with a given index node is a content table covering only those keyword occurrences which are referenced by this index node, i.e. reached by its label path. In the case of the index node #5, e.g., the corresponding subtable maps the keyword *"index"* to the document nodes &5 and &8, which are exactly the nodes which are reached by #5's label path, /book/chapter/section/paragraph, and which contain the keyword *"index"*. Analogously, *"XML"* is mapped to the document node &8. From the fact that the subtable does not contain any other keywords, one can deduce that only these two occur in document nodes referenced by index node #5.

3.2.2.2 Retrieval algorithm

Compared to the content-centric approach, where content-aware navigation is achieved simply by choosing the appropriate subindex before path matching, the structure-centric approach is slightly more complicated. Since there is merely one navigational structure to be used for query



(a) structure-centric navigational structure (annotated)



(b) structure-centric content/annotation table

Figure 3.3: Data structures of the structure-centric approach to content-aware indexing

evaluation, the query keywords to be retrieved must be used during navigation to prune irrelevant index paths. Obviously, the necessary keyword information is stored in the subtables attached to all index nodes. They reveal which keywords occur in the document nodes referenced by the respective index nodes. However, for effective pruning one needs to assess the relevance not only of the index node whose subtable is being examined, but also of its descendants without first visiting their subtables. For instance, when evaluating the query `//*/paragraph["XML"]`, as soon as the keyword information for the index node #0 is available it must be clear that the path to index node #2 can safely be ignored, since it does not reference document nodes where the keyword *"XML"* occurs. How exactly this is achieved in the structure-centric approach is discussed in the next subsection (see page 35). Suffice it to say here that each index node bears sufficient information to decide whether any of its descendants references document nodes with the right keywords.[14] This test will be called the *navigation relevance check* hereafter. But not only the index tree, also every query tree is modified to support the relevance check. In much the same way as the index tree is supplied with keyword information at indexing time for subsequent relevance checks, the query tree is prepared at evaluation time in a simple preprocessing step to be explained later. Since the preprocessing takes place before path matching, it is referred to as the retrieval phase 0 in the remainder of this work.

```
1  // recursively matches a label path with query keywords against the index tree, starting from a given node
2  // returns the set of document nodes reached by the given label path where the given keywords occur
3  procedure processQueryPathCADG (LabelPath p, Set K, IndexNode i)
4
5     // retrieval phase 0: query preprocessing
6     if first incarnation then
7        call preprocessQuery (p, K)
8     end if
9
10    // retrieval phase 1: content-aware navigation
11    if p has more edges to be matched then
12       l := the label of p's first edge
13       if i has a child i_l reached by an edge labelled l and isRelevantNav(i, K) then
14          p' := p with the first edge chopped off
15          return processQueryPathCADG (p', K, i_l)
16       else
17          return ∅
18       end if
19    end if
20
21    // retrieval phase 2: content-aware occurrence fetching
22    if isRelevantOcc(i, K) then
23       return cadg_sc(i, K)
24    else
25       return ∅
26    end if
27
28 end procedure
```

Algorithm 3.1: Retrieval with the CADG (simplified)

The listing 3.1 summarizes the CADG's retrieval algorithm for query paths on a high level of abstraction. Three index-specific methods are called which will be explained further below, when concrete realizations of the CADG are discussed. The *preprocessQuery*() method (line 7) prepares the query tree for content-aware evaluation. Note that it is executed only once. Navigation

[14] In figure 3.3, every index node has the appropriate content information written next to it for convenience.

is made content-aware by calling the *isRelevantNav*() method (line 13), which implements the *navigation relevance check*. Similarly, *isRelevantOcc*() in line 22 avoids database look-ups for irrelevant index nodes, implementing the *occurrence-fetching relevance check* (see section 3.2.3). The structure/content join being part of the function $cadg_{sc}$ (see above), retrieval phase 3 has been dropped.

Based on the aforementioned premises, the retrieval algorithm is straightforward. After the query has been preprocessed, path matching takes place just like with the ordinary DataGuide, except that index nodes for which the relevance check fails are ignored during navigation, along with their entire subtree. Occurrence fetching for the leaves of the matching index paths is performed as described in section 3.2.1.2 for the content-centric approach. For instance, consider the query above and the index structure from figure 3.3 *(a)*. The first node to be considered is #0. The relevance check for the query keyword *"XML"* succeeds because one of #0's descendants, #5, references a document node where this keyword occurs. Again, this is known from the keyword information attached to #0, which will be described soon. Proceeding with the path matching for $//*/$paragraph, the index node #1 is reached. Before accessing its child node #2, the relevance check for #1 fails as the only two keywords to be reached in this branch are *"index"* and *"survey"*. Hence the index node #1 along with all its descendants, i.e. node #2 in this case, is discarded, and path matching continues with its sibling, #3, and then #4. Both pass the relevance check for the same reason as their ancestor #0. Finally, the index node #5 is reached. Of course its relevance check succeeds, too, and since it is also a structural match, #5's occurrences for the keyword *"XML"* are fetched from the content/annotation table. The resulting set of document nodes contains &8 as its only element, as expected. The last index node, #6, is treated like #1 before.

3.2.2.3 Storage management

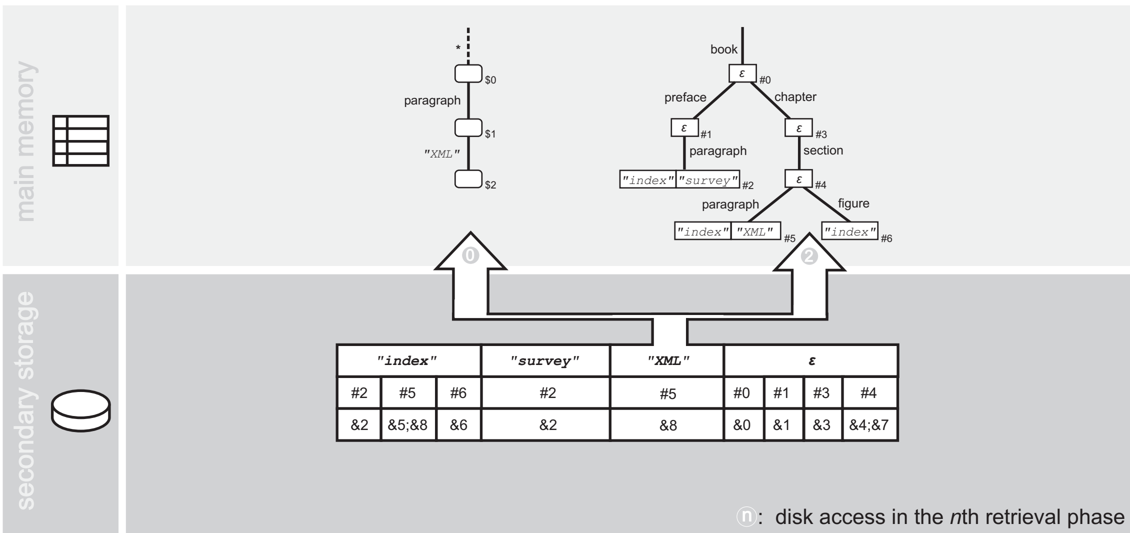


Figure 3.4: Storage management of the structure-centric approach to content-aware indexing

Figure 3.4 illustrates how the structure-centric approach re-establishes the separation between a main-memory resident navigational structure and disk-resident content and annotation data, as known from the DataGuide (see figure 2.3 on page 21). Compare this to the content-centric approach shown in figure 3.2 on page 29, where all keyword-specific navigational structures are kept on secondary storage devices until selected ones are loaded into main memory at query time. The content/annotation table is stored on disk in both cases. However, in the case of the structure-centric approach each index node keeps the keyword information provided by the corresponding content/annotation subtable, as symbolized in figure 3.4: the keyword columns' headers are still attached to the index nodes, while the respective lists of document node IDs are stored in the full content/annotation table on the disk. The table itself is accessed on two occasions: first during query preprocessing (retrieval phase 0), and once more during occurrence fetching (retrieval phase 2). Strictly speaking, the access in retrieval phase 0 is not necessary, as discussed in section 4.2.3 which introduces a pure main-memory implementation of the query preprocessing procedure.

3.2.2.4 Discussion

Obviously the index structure presented in this subsection satisfies the objectives from section 3.1, avoiding needless disk operations, unnecessary index navigation, and content/structure joins based on document node IDs. The evaluation overhead compared to the ordinary DataGuide, caused by query preprocessing, is relatively modest, although one can expect that in the case of a very simple query inducing hardly any navigation, the DataGuide might outperform the CADG due to this additional disk operation. To reduce the evaluation overhead of the CADG, section 3.2.3.2 below introduces an efficient heuristic preprocessing method based on keyword signatures.

Besides, the structure-centric approach addresses the disadvantages of the content-centric approach by using a single, main-memory resident navigational structure. Its concept of content-aware navigation is adapted to this feature, being based on a relevance check for index nodes at query time. Note that in addition to those index nodes which are relevant from the content-matching point of view, and whose relevance check therefore succeeds, the retrieval procedure visits some irrelevant nodes just to find out that their branch can be skipped. As shown in the example above, an irrelevant index subtree can only be pruned after the relevance test has been carried out for its root, and failed. On the one hand, as far as content-aware navigation is concerned, this makes the structure-centric approach slightly less efficient than the content-centric one where irrelevant index nodes are never touched during query evaluation. The reason is that the former relies on the relevance check to prune them at query time, whereas the latter excludes them from navigation straight away by pruning at indexing time. On the other hand, the overhead for the relevance check (a main-memory operation) is neglectable compared to the overhead for index loading (a secondary-storage operation) incurred by the content-centric approach. Moreover, it can be further reduced when heuristic instead of exact content-awareness is acceptable, as explained in the next subsection. Finally, the structure-centric approach handles keyword con- and disjunctions without a substantial loss of retrieval performance, unlike the content-centric approach.

As far as storage is concerned, the content-centric navigational structure is usually more redundant than the structure-centric one, duplicating index paths as explained in section 3.2.1.4. Apart from the substantial storage overhead which might result from this redundancy, especially when indexing document collections with many different keywords per node, query performance is affected because different navigational structures need to be fetched from disk at query time, as mentioned above. By contrast, the space consumption of the content/annotation table is equal in both cases. Summing up, one can say that while both the content-centric and the structure-centric approach introduce a certain redundancy compared to the ordinary DataGuide (as observed also in chapter 5), the latter takes greater advantage of it, its redundant data structures residing on disk during the whole retrieval process.

3.2.3 Realizations of the CADG

The previous subsection pointed out that the structure-centric approach to content-aware indexing is superior to the content-centric approach, due to its efficient storage management.[15] However, the

[15] As anticipated in the introduction to section 3.2, the content-centric approach, serving as a starting point only to motivate and contrast with the structure-centric approach, is not further pursued.

price to pay for this efficiency is the relevance check mentioned in section 3.2.2.2, which allows for content-aware navigation in a navigational structure which is not keyword-specific per se. Recall that the relevance check for a given index node and keyword tests whether or not this index node or any of its descendants references at least one document node where the keyword in question occurs. When it fails, the index node being checked can be skipped during retrieval, along with all its descendants, since none of them will ever contribute any occurrence of the requested keyword to the query result.

So far, the relevance check has been treated solely as a means to enhance retrieval phase 1 (navigation), allowing for content-aware navigation in an otherwise ordinary DataGuide. Yet it also saves needless disk operations during retrieval phase 2 (occurrence fetching). Before fetching keyword occurrences for an index node matching the query path and the keywords contained in this path's textual leaf, the relevance check can tell whether this index node actually references relevant document nodes. This *occurrence-fetching relevance check* is stricter than the *navigation relevance check* described before, because the checked index node itself is required to reference relevant document nodes in order to pass the test, while relevant descendant index nodes are ignored.

It has already been mentioned that certain modifications of both the index tree and the query tree are necessary to make an ordinary DataGuide content-aware. This subsection deals with two alternative methods to realize the relevance check (both for navigation or occurrence fetching), including the modifications of the navigational structure and the additional query preprocessing step (retrieval phase 0). The two corresponding index structures, the *ID-Comparison CADG* on the one hand and the *Signature CADG* on the other hand, are the core contributions of this work. Both have been implemented and tested, as documented in chapter 4 and 5, respectively.

3.2.3.1 ID-Comparison CADG

Overview. One way to decide whether occurrences of a given keyword are referenced by any index node in a specific subtree of the navigational structure is to collect all index nodes referencing document nodes where the keyword occurs, and then to determine whether any of these index nodes is part of the subtree being checked. If this is true, the relevance check succeeds, otherwise it fails. The set of index nodes referencing relevant document nodes is easily determined by looking up the IDs of all index nodes associated with the desired keyword in the content/annotation table. If there are no such index nodes, the query can be immediately discarded as unsatisfiable. For instance, reconsider the index structure shown in figure 3.3 on page 32. As discussed in section 3.2.2.2, the relevance check for the keyword *"XML"* should fail when performed for index node #1, whereas for #3 it is supposed to succeed. A look-up in the content-annotation table shown at the bottom of figure 3.3 *(b)* reveals that the set of index nodes referencing document nodes where *"XML"* occurs has a single member, #5. Obviously the index node #5 is not part of the subtree rooted at #1, such that for #1 the relevance check just proposed fails indeed. By contrast, #5 is a descendant of the index node #3, which consequently passes the relevance check for the keyword *"XML"*.

The above procedure involves an ancestor/descendant test for index nodes, which is needed to decide whether a given index node is part of a given subtree of the navigational structure. To this end, the path leading to the index node in question is traversed upwards until either the root of the subtree is reached, in which case the ancestor/descendant test succeeds, or the root of the navigational structure, in which case the test fails (unless the subtree root and the index root are identical). Section 3.3.3 below puts forward an optimization which saves this additional index navigation.

Navigational structure. The relevance check of the ID-Comparison CADG relies on index node IDs to determine whether or not a path in the navigational structure can be pruned. Since a CADG's index nodes are supposed to have unique IDs for occurrence fetching anyway, they need not be especially adapted to the ID-based relevance check. Yet there must be a means to get hold of any index node by its ID, provided the ancestor/descendant test involves upward navigation, as described in the previous paragraph. With the optimization from section 3.3.3, however, this ID-based index node access is not necessary.

Figure 3.5 illustrates the navigational structure (without annotations) of the ID-Comparison Content-Aware DataGuide for the document collection shown in figure 2.2 *(b)* on page 17. Apart from their IDs, the index nodes lack any further content information. The navigational structure is therefore identical to the ordinary DataGuide's in figure 2.2 *(e)*.

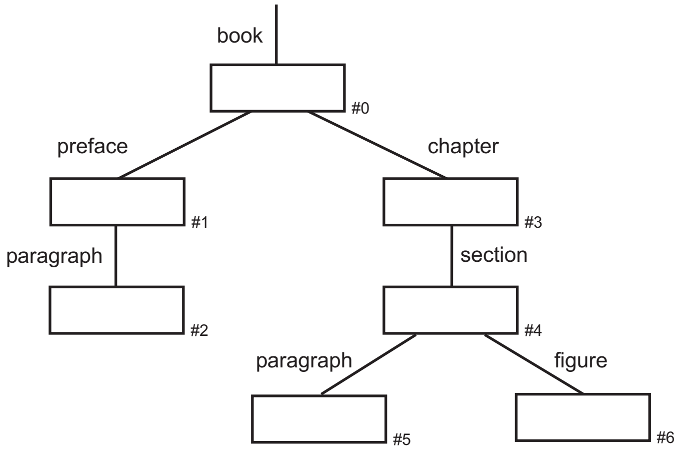


Figure 3.5: Navigational structure of the ID-Comparison CADG

Query preprocessing. The adaptation of a query tree to the ID-based relevance check is more complex than that of the index tree. While the procedure sketched in the overview above is straightforward for a single query keyword, it does not handle cases in which multiple keywords are conjoined or disjoined or, more prominently, in which a query node has multiple children each with its own query keywords. The latter case refers to a non-trivial query tree, whereas the former applies also to a single query path. To have both of them supported by the ID-based relevance check, a given query tree is preprocessed in a recursive manner, the recursion anchors being textual query nodes. The following procedure assigns to each query node q a set I_q of sets of index node IDs. This second-order set is used during the relevance check (whether for navigation or occurrence fetching) of any index node matching q, as described further below.

Textual query nodes. Conceptually, a textual query node q_t with a single keyword k_0 is associated with the set I_{k_0} of IDs of those index nodes referencing document nodes where k_0 occurs. For technical reasons which become apparent in the next paragraph, we use $I_{q_t} := \{I_{k_0}\}$ instead of just $I_{q_t} := I_{k_0}$. I_{k_0} is computed from a look-up for k_0 in the content/annotation table as mentioned in the overview above. If the query node represents a conjunction $\bigwedge_{u=0}^{p} k_u$ of multiple keywords, their respective sets I_{k_u} are intersected, $I_{q_t} := \{\bigcap_{u=0}^{p} I_{k_u}\}$, because according to the semantics of the tree query formalism (see section 2.3) the conjoined keywords must all occur in the same document node, and hence be referenced by the same index node for the query to match. To illustrate this point, consider a navigation relevance check[16] for an index node i and the keyword conjunction $\bigwedge_{u=0}^{p} k_u$. If during the check any of the index nodes in the intersection $\bigcap_{u=0}^{p} I_{k_u}$ turns out to be a descendant of the index node being checked, then the subtree rooted at i is guaranteed to contain an index node referencing a document node

[16] The occurrence-fetching relevance check is covered further below.

where all keywords k_u, $0 \leq u \leq p$, occur—which justifies that the relevance check succeeds. Analogously, a query node representing a disjunction $\bigvee_{u=0}^{p} k_u$ of keywords is associated with the union $I_{q_t} := \{\bigcup_{u=0}^{p} I_{k_u}\}$ of sets of relevant index node IDs. If $I_{q_t} = \{\emptyset\}$, the query is rejected as unsatisfiable (without entering retrieval phase 1), because no index node matching the query path references any document nodes where the desired keywords occur.[17]

Structural query nodes. As specified in section 2.3, the children q_v, $0 \leq v \leq m$, of a structural query node q_s are supposed to be conjoined, i.e. there must be a matching document node for each of them. However, there is a subtle but important difference between the conjunction of a structural query node's children and the conjunction of a textual query node's keywords, which affects the preprocessing procedure of the ID-Comparison CADG. It is the reason why second-order sets have been used for query node in the previous paragraph. For example, in the case of a navigation relevance check, if first-order sets of relevant index node IDs I_{q_v} were intersected, like the sets I_{k_u} in the keyword conjunction above, the check for an index node i and the query keywords from *one* of the paths leaving q_s would only succeed if there were a descendant of i referencing a document node where *all* of the query keywords below q_s occur. This clearly violates the query semantics. Instead q_s is associated with all ID sets I_{q_v} simultaneously, i.e. with $I_{q_s} := \{I_{q_0}, \ldots, I_{q_m}\}$. The child nodes' ID sets I_{q_v} are processed separately during the relevance check (see the next paragraph). This way the check succeeds as soon as there exists for each child query node q_v one descendant of i referencing a relevant document node, without demanding that it be the same for all of them. In case q_s has no children, $I_{q_s} := \emptyset$ is used as a "dont't care" symbol, ensuring that the navigation relevance check for such query nodes always succeeds. Note that all children q_v are treated alike, regardless of whether they are structural or textual query nodes. As a consequence, the preprocessing also copes with mixed-content query trees.

Figure 3.6 illustrates the query `/book[./preface/paragraph["index"] and .//*["XML"]]` as a query tree preprocessed for the ID-Comparison CADG. Each row in a query node q contains a set of relevant index node IDs, which is an element of the query node's second order set I_q. For instance, the root of the query tree, `$0`, is associated with the set $I_{\$0} = \{\{\#2; \#5; \#6\}; \{\#5\}\}$. The different index node ID sets associated with the query nodes have been computed using the content/annotation table shown in figure 3.3 *(b)*. Compare the resulting query tree to the original one, which is shown in figure 2.1 *(a)* on page 13.

Relevance check. First the navigation relevance check is considered. As described in section 3.2.2.2, the check is performed whenever an index node is reached during retrieval phase 1. From the path matching procedure given in algorithm 2.2 on page 20, it is clear that there is always a fixed structural query node q_s to be matched while navigating the index tree. Hence whenever an index node i is reached during path matching, the navigation relevance check is carried out for i and q_s. After query processing, the set I_{q_s} of index node ID sets associated with q_s contains information about all keywords in the subtree rooted at q_s (see the previous paragraph). This query keyword information is used for i's relevance check as follows. In each set $I_{q_v} \in I_{q_s}$, a descendant of i is searched.[18] The navigation relevance check for i succeeds if and only if there is at least one descendant in each set I_{q_v}. Note that as a special case, the relevance check succeeds if $I_{q_s} = \emptyset$.

The occurrence-fetching relevance check is performed when processing a textual query node during retrieval phase 2. It takes place for every index node matching the query node's parent, provided its navigation relevance check succeeded. For instance, let q_t be a textual query node, and i an index node matching the query path from the query tree's root down to q_t's parent node, q_s. When q_t is processed, i has already matched q_s. Assume the navigation relevance check

[17] Section 2.3 requires every textual query node to contain at least on keyword. However, this is a mere convention. If empty textual query nodes were allowed, one could simply set $I_{q_t} := \emptyset$, which causes q_t to be treated like a childless structural query node (see the next paragraph).

[18] The search need not be performed sequentially, as mentioned in section 4.4.

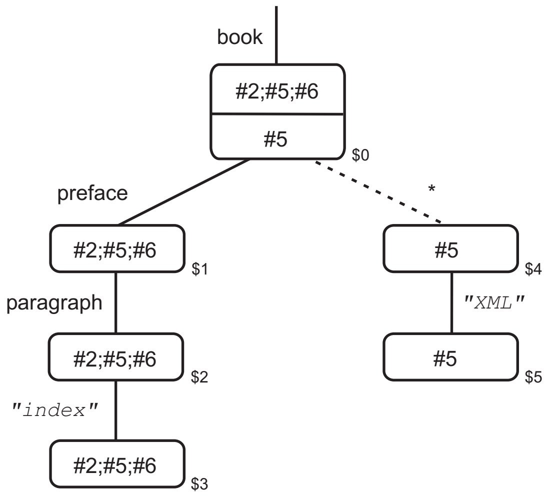


Figure 3.6: Query preprocessing with the ID-Comparison CADG

for i and q_s succeeded, which means that either i or any of its descendants references relevant document nodes, or both. Then an occurrence-fetching relevance check is performed for i and q_t. To determine whether occurrence fetching for this index node is justified, i is searched in the only member I_k of the singleton set I_{q_t}, which contains the index nodes relevant for q_t's keywords.[18] The occurrence-fetching relevance check for i succeeds if and only if $i \in I_k$.

As stated above, this second procedure is similar to the one for the navigation relevance check, except that it is based on an identity relation between the index node being examined and the members of I_q, rather than an ancestor/descendant relation. This explains why the occurrence-fetching relevance check is indeed stricter than the navigation relevance check. In fact, a success of the former implies a success of the latter for a given index and query node, but not vice versa. This reflects the intended meaning of the respective relevance checks, as can be easily verified.

Retrieval example. Consider the navigational structure shown in figure 3.5 and the preprocessed query tree from figure 3.6. Beginning with the label book, path matching identifies the index node #0 as a structural match for the query node $0. #0's relevance check succeeds because in both sets associated with $0, {#2; #5; #6} and {#5}, there is a descendant of #0 (namely #5). $0 has two structural children, which are processed one after the other. First, the preface label linking $0 to $1 in the query tree leads from #0 to #1 in the index tree. The following navigation relevance check for #1 succeeds because one member of $1's ID set {#2; #5; #6}, #2, is a descendant of #1. Path matching continues along the edge labelled paragraph reaching $2. The only possible structural match for $2 is #2, whose navigation relevance check succeeds because #2 is also its own descendant (remember that the descendant relation is reflexive, as specified in section 2.3). Obviously this implies that $2 passes the occurrence-fetching relevance check for $3, being a member of $3's ID set {#2; #5; #6} which is the same as for $2. Hence occurrence fetching for #2 and $3 is guaranteed to yield at least one document node where $3's keyword, *"index"*, occurs. From the content/annotation table in figure 3.3 *(b)*, one can see that there is indeed one such document node, &2. Obviously, no other index path matches this branch of the query tree.

The second structural child of $0, $4, is reached by a soft edge without any label constraint. Structural matches for $4 are therefore the nodes in the index subtree rooted at #0, i.e. all index nodes. #0 as an ancestor of #5, the singleton member of $4's only node ID set {#5}, passes the navigation relevance test for $4, but immediately fails in the occurrence-fetching relevance check for $5 because it is not itself a member of this set. The next node to be checked, #1, is already ruled out by the navigation relevance check for $4: after all #5 is not part of the subtree rooted at #1. As a consequence, the left part of the navigational structure in figure 3.3 *(a)*, comprising #1 and #2, is pruned off, i.e. the latter never even enters retrieval phase 1. By contrast, #0's second child #3 passes the navigation relevance check for $4, being an ancestor of #5. Since there are no more structural query nodes to be processed in this path, #3 undergoes the occurrence-fetching relevance test for $5, which fails because #3 is not a member of $5's ID list {#5}. Note that this does not cause #3's subtree to be pruned: while the failure in the occurrence-fetching relevance check reveals that #3 itself does not reference document nodes where $4's keyword *"XML"* occurs, the successful navigation relevance check asserts that one of #3's descendants does. There are three more structural matches for $4, namely #4, #5, and #6. The first candidate passes the navigation relevance check for the same reason as its parent, #3. Analogously, it fails in the occurrence-fetching relevance check. The second candidate, #5, passes the both tests, being included in both $4's and $5's set of relevant index nodes. Occurrence fetching for #5 yields &8 as the only hit for $4. Finally the third candidate for this query node, #6, is examined. Obviously it fails in the navigation relevance check. Just like #1 for the keyword *"index"* before, it is excluded (along with its descendants, if there were any) from further navigation and occurrence fetching.

The resulting hits for the entire query tree are shown as a Complete Answer Aggregate in figure 3.7. Compare this to the original document tree in figure 2.2 on page 17.

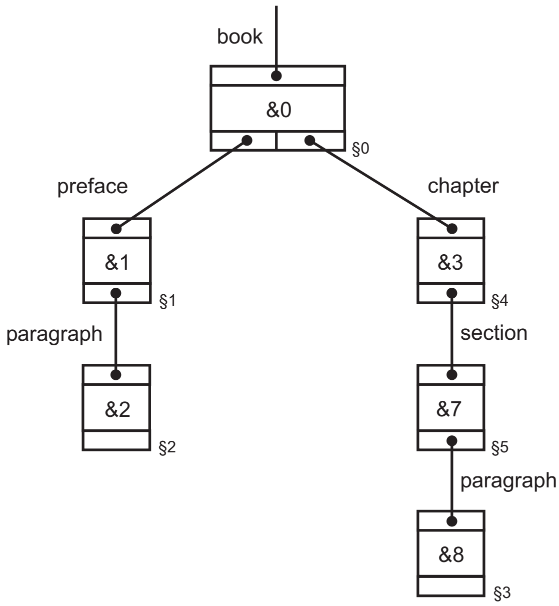


Figure 3.7: CAA holding hits retrieved by the ID-Comparison CADG

3.2.3.2 Signature CADG

Overview. Content-aware navigation and occurrence fetching based on index node IDs, as described in the previous subsection, entails set operations like intersection, union, and element test during the retrieval phases 0, 1, and 2. Depending on the underlying navigational structure as well as the query keywords, this may cause a significant overhead at query time. The alternative presented in this subsection therefore strives for a lightweight relevance check which is affordable even in the worst case, when no paths can be pruned and the relevance check could just as well have been omitted. The idea is to represent the keyword information about every index node and its descendants as a signature (see section 2.5 for a short introduction to this common Information

Retrieval technique). As mentioned before, signatures are designed for concise representation and fast processing of heuristic information. As a consequence, the retrieval overhead of the Signature CADG is often reduced considerably compared to the ID-Comparison CADG (see section 5.4.2.3 for an empirical comparison of both). Alas, signature-based navigation and occurrence fetching is inherently inexact, which means that the approach cannot guarantee to exploit all possible pruning opportunities. The ID-Comparison CADG has the advantage of being exact, i.e. optimal in this respect. However, pruning precision on the one hand and retrieval precision on the other hand must not be confused. The Signature CADG, while employing a heuristic means of content-aware navigation and occurrence fetching, is still guaranteed to retrieve exact query results. In terms of classic Information Retrieval, it features optimal retrieval precision and recall, just like the ID-Comparison CADG.

Both the indexed keywords occurring in document nodes and the query keywords attached to textual query nodes are represented as signatures. These *keyword signatures* are fetched during retrieval phase 0. For the navigation relevance check in phases 1 and 2, *index node signatures* or *query node signatures* are created from the keyword signatures associated with a document node or textual query node, respectively, as explained in the next paragraphs. The result are *signature hierarchies* built over both the index and the query tree. A similar, although different technique has been used in an earlier approach, the Signature File Hierarchy (see section 6.3 for details).

Navigational structure. Every index node of the Signature CADG has two index node signatures, the *occurrence-fetching signature* and the *navigation signature*. As its name suggests, the former, representing all keywords occurring in the document nodes referenced by this index node, is used for the occurrence-fetching relevance check. The latter, by contrast, is meant to support the navigation relevance check. It is built over all keywords occurring in the document nodes referenced anywhere in the index node's subtree. In other words, while the occurrence-fetching signature of an index node represents heuristic content information about all document nodes reached by the same label path as this index node, its navigation signature additionally includes all document nodes whose label path has the index node's path as a prefix only. Both signatures can be created simultaneously at indexing time. The following procedure, which is performed for every keyword occurrence being indexed, assumes that all keywords occurring in the document collection have been assigned a keyword signature, as explained in section 2.5. Besides, each index node's navigation and occurrence-fetching signatures are supposed to have all bits set to `0` initially.

Occurrence-fetching signatures. Let s_k be the signature for the keyword whose occurrence in the document node d is being indexed, and s_o the occurrence-fetching signature of the index node i referencing d. First of all, s_o must be updated to reflect the new keyword occurrence. To this end, s_o is merged with s_k in a bitwise disjunction, $s_o := s_o \sqcup s_k$. Note that while this operation leaves s_o unchanged when the same keyword is indexed repeatedly, multiple different keywords may cause it to saturate, i.e. to have all bits set to `1` eventually. This effect is discussed further below.

Navigation signatures. Just like s_o, i's navigation signature s_n is disjoined with s_k, i.e. $s_n := s_n \sqcup s_k$. However, since i's ancestors need to reflect the new keyword occurrence as well, all navigation signatures along the path from the index root to i are treated in the same way: $s_{n,u} := s_{n,u} \sqcup s_k$, for all ancestors i_u of i with a navigation signature $s_{n,u}$.

Figure 3.8 illustrates the navigational structure (without annotations) of the Signature Content-Aware DataGuide for the document collection shown in figure 2.2 *(b)* on page 17. Each index node has two signatures, a navigation signature (top) and an occurrence-fetching signature (bottom). Contrast this with the ID-Comparison CADG's navigational structure in figure 3.5, which does without explicit content information. The assumed signatures for the indexed keywords *"index"*, *"survey"*, and *"XML"* are inset in the lower left corner of the figure.

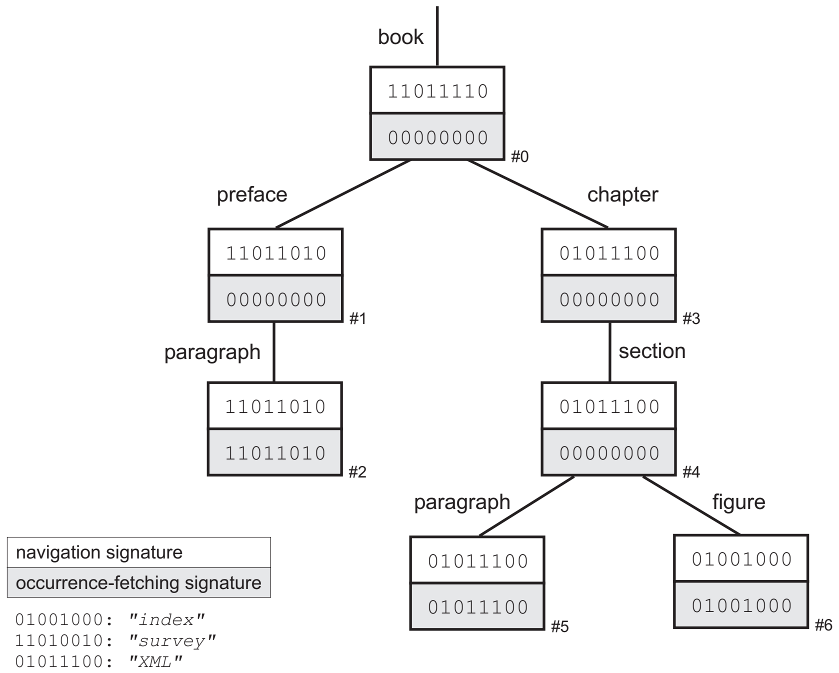


Figure 3.8: Navigational structure of the Signature CADG

Query preprocessing. Preparing the query tree for content-aware navigation and occurrence fetching based on signatures is similar, although even simpler than for the index tree. Every structural query node q_s has a single signature s_{q_s}, comparable to the navigation signatures of index nodes, that indicates which keywords are contained in the textual query nodes below q_s. A textual query node's signature s_{q_t} represents the query keywords attached to this node.[19] The keyword signatures are either fetched from a signature table, where they have been stored at indexing time, or created on the fly.[20] The procedure for creating a signatures hierarchy for the query tree during retrieval phase 0 resembles to a certain degree the one for the ID-Comparison CADG described in the previous subsection:

Textual query nodes. The signature s_{q_t} associated with a textual query node q_t is created from the keyword signatures s_{k_u} of all keywords k_u attached to this query node $(0 \leq u \leq p)$. If there is only one such keyword, say k_0, then $s_{q_t} := s_{k_0}$.[21] In the case of a keyword conjunction $\bigwedge_{u=0}^{p} k_u$, s_{q_t} is set to be the bitwise disjunction of the keyword signatures, $s_{q_t} := \bigsqcup_{u=0}^{p} s_{k_u}$. The reason for this decision, which may seem counter-intuitive at first sight, will become apparent when examining the occurrence-fetching relevance check. Informally, disjoining the signatures guarantees that each keyword "leaves its trace" in the query node's signature, to stick to the metaphor from section 2.5. Analogously, $s_{q_t} := \bigsqcap_{u=0}^{p} s_{k_u}$ for a keyword disjunction $\bigvee_{u=0}^{p} k_u$ in q_t.

Structural query nodes. Since the approach to content-awareness described here, which is based signatures instead of index node ID sets like the ID-Comparison CADG, does not involve

[19] Obviously, query nodes do not need a second signature because there is no occurrence fetching for query nodes.
[20] The issue of signature creation is covered in section 4.2.3.
[21] As specified in section 2.3, a textual query node is required to contain at least one keyword by convention. However, a textual query node without keywords, if it were needed for some reason, could simply be assigned a signature consisting entirely of unset bits, which would ensure that any index node would match in retrieval phase 2.

ancestor/descendant tests on index nodes, the conjunction of multiple children of a structural query node q_s can be handled just like the conjunction of multiple keywords in a textual query node. In other words, if q_s has children q_v, $0 \leq v \leq m$, each with its own signature s_{q_v}, then $s_{q_s} := \bigsqcup_{v=0}^{m} s_{q_v}$. The signature of a childless structural query node is left unchanged, i.e. it has no bit set. Thus any index node's navigation relevance check will succeed, as one would expect for a query node without textual constraints.

Figure 3.9 illustrates the query `/book[./preface/paragraph["index"] and .// * ["XML"]]` as a query tree preprocessed for the Signature CADG. Contrast this query tree with the one in figure 3.6, where each node's content information is represented by one or more sets of index node IDs rather than signatures. The signatures for the query keywords *"index"* and *"XML"* are taken from figure 3.8 above. Note how the signature of query node `$1` is "absorbed" by the one of query node `$4`, which happens to have the same plus two additional bits set. The resulting merged signature, attached to query node `$0`, is identical to `$4`'s.

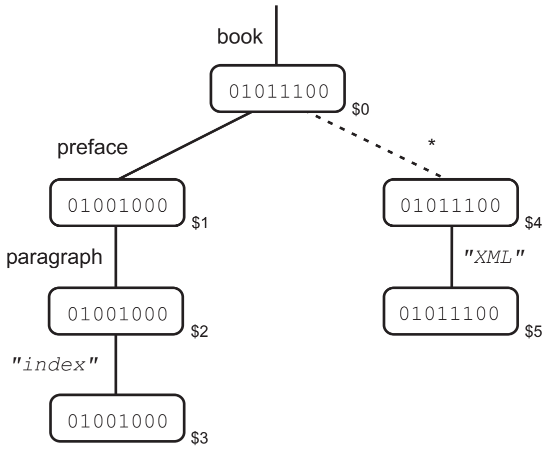


Figure 3.9: Query preprocessing with the Signature CADG

Relevance check. The navigation relevance check for an index node i and a structural query node q_s simply consists of a bitwise implication of q_s's signature and i's navigation signature, $s_{q_s} \sqsubset s_n$ (see section 2.5 for the definition of $\sqsubset$). This means that those bits which are set in s_{q_s} because of the query keywords below q_s must also be set in s_n, which is definitely the case when every query keyword occurs in some document node referenced by i or its descendants. However, the inverse is not always true: $s_{q_s} \sqsubset s_n$ may also hold when the keywords in q_s's subtree do not all occur in the document nodes referenced in i's subtree. Recall from section 2.5 how the merging of different sets of keyword signatures can produce identical results. Likewise, other keywords than the ones which are responsible for s_{q_s} can make s_n look as if i were relevant, although it is not. In such a case, path matching continues in i's subtree of the navigational structure, ignoring that occurrence fetching for any of its index nodes is doomed to fail.

Analogously to the navigation relevance check just described, the occurrence-fetching relevance check for an index node i and a textual query node q_t is just the bitwise implication of q_t's signature and i's occurrence-fetching signature, $s_{q_t} \sqsubset s_o$. It succeeds when q_t's keywords occur in document nodes referenced by i itself (regardless of its descendants), where they cause the same bits to be set in s_o as in s_{q_t}. But like for the navigation relevance check, i may also pass the check when the query keywords do not occur in its document nodes, in which case occurrence fetching for this index node (including a database access) is performed in vain. The next paragraph gives an example of this phenomenon.

Retrieval example. In the following the query depicted in figure 3.9 is evaluated using the Signature CADG in figure 3.8, with the content/annotation table in figure 3.3 *(b)*. This setting corresponds to the retrieval example for the ID-Comparison CADG in section 3.2.3.1. The query tree's root label book leads to index node #0, with the navigation signature $\boxed{\texttt{11011110}}$. The navigation relevance check for #0 proves the bitwise implication of this signature by $0's, $\boxed{\texttt{01011100}} \sqsubset \boxed{\texttt{11011110}}$. Hence path matching continues, starting with the left branch of the query tree. The preface edge leads to #1 in the index tree, and then paragraph reaches #2. Both index nodes having identical navigation signatures, just like the two corresponding query nodes $1 and $2, the navigation relevance check in both cases is the same, namely $\boxed{\texttt{01001000}} \sqsubset \boxed{\texttt{11011010}}$, which again succeeds. $2's only textual child node $3 triggers the occurrence-fetching relevance check for #2, which tests $\boxed{\texttt{01001000}} \sqsubset \boxed{\texttt{11011010}}$ once more—because $3's signature is identical to $2's, and #2's occurrence-fetching signature equals its navigation signature (as is always the case for leaves in the index tree). The successful occurrence-fetching check is followed by a look-up in the content/annotation table (see figure 3.3 *(b)*), which yields &2 as the only occurrence of the keyword *"index"* under the label path /book/preface/paragraph. Path matching for this branch of the query tree comes to an end with the structural mismatch of the only remaining child of #0, which is reached by a chapter instead of a preface edge.

The second path in the query tree, /book//$*$["XML"], is evaluated as follows. The first step after /book is matched by all index nodes, as observed in section 3.2.3.1. #0 passes the navigation relevance check for $4, $\boxed{\texttt{01011100}} \sqsubset \boxed{\texttt{11011110}}$, but fails in the occurrence-fetching relevance check $\boxed{\texttt{01011100}} \sqsubset \boxed{\texttt{00000000}}$ by lack of keywords. The index root's left child #1 immediately fails in the navigation relevance check for $4, $\boxed{\texttt{01011100}} \sqsubset \boxed{\texttt{11011010}}$, since the antepenultimate bit is set in $4's signature, but not in #1's. Hence the left branch of the index tree is pruned, and path matching continues with #3. It passes the navigation relevance check for $4, $\boxed{\texttt{01011100}} \sqsubset \boxed{\texttt{01011100}}$, but not the occurrence-fetching relevance check for $5, $\boxed{\texttt{01011100}} \sqsubset \boxed{\texttt{00000000}}$ (like in the case of #0 above, the second signature indicates that #3 does not reference any document node with textual content). The same is true for #3's only child #4. The two remaining index nodes, however, behave differently: while #5 passes both the navigation relevance test for $4 (same operation as above) and the occurrence-fetching relevance check for $5 ($\boxed{\texttt{01011100}} \sqsubset \boxed{\texttt{01011100}}$), contributing the document node &8 to the result, its sibling #6 fails already in the first check, $\boxed{\texttt{01011100}} \sqsubset \boxed{\texttt{01001000}}$ (the fourth and sixth bits are missing in the second signature). Accordingly, #6 is excluded from further navigation (which cannot take place anyway, #6 being a leaf node) and occurrence fetching, thus saving a look-up in the content/annotation table.

Note that if another keyword with a suitable signature occurred in the document node referenced by #6's, e.g. the keyword *"query"* with the signature $\boxed{\texttt{00011100}}$, then #6 would be mistaken to be relevant for $5's keyword *"XML"*. The reason is that both #6's navigation and occurrence-fetching signature would then be the bitwise disjunction of the two signatures representing *"index"* and *"query"*, $\boxed{\texttt{01001000}} \sqcup \boxed{\texttt{00011100}} = \boxed{\texttt{01011100}}$, which equals the keyword signature for *"XML"*. Hence #6 would pass the navigation relevance check for $4 and the occurrence-fetching relevance check for $5, just like its sibling #5. Only after the look-up in the content/annotation table (i.e. after an expensive database access) would #6 turn out to be a false hit. This illustrates how the Signature CADG trades pruning precision off against navigation efficiency. But in either case, the query results are the same as for the ID-Comparison CADG (see figure 3.7 on page 40) since the Signature CADG combines heuristic pruning with exact retrieval, as pointed out above.

3.3 Optimizations

Several optimization techniques can be applied to the CADG as well as the ordinary DataGuide to enhance individual retrieval phases. The four techniques presented in this section include two which are known from the literature and two new ones developed in the course of this work. All four have been implemented and tested with the DataGuide, ID-Comparison CADG, and Signature CADG (see chapter 5). As discussed in chapter 6, they can also be applied to some related indexing approaches.

3.3.1 Indexing of indirect keyword occurrences

Overview. A common feature of query languages for Semi-Structured Data are operators for evaluating partially specified query paths, whose matching document paths may contain nodes which do not correspond to any node in the query path. The formalism introduced in section 2.3, e.g., provides so-called soft edges for both structural and textual query nodes. In the path query language XPath, soft-edge queries can be simulated using the descendant operator // (see sections 3.2.3.1 and 3.2.3.2 for an example). When used with structural query nodes, this construct expresses a closure of the parent/child relationship on document nodes.[22] Accordingly, path matching with soft edges leading to structural query nodes is normally realized as an exhaustive traversal of a subtree in the navigational structure, which serves to construct the set of ancestor/descendant pairs recursively in parent/child steps. By contrast, soft edges to textual query nodes cannot be regarded as a closure construct because textual query nodes, being leaves by definition, must not be chained together. As described in section 2.3, soft-edged textual query nodes still trigger an exhaustive search, just like structural nodes, and a separate occurrence fetch for each structural match found during navigation. In terms of the two occurrence definitions from section 2.3, the indirect keyword occurrences to be retrieved are computed from the indexed direct occurrences at query time, which makes the retrieval phases 1 and 2 much more complex.

The idea of this optimization is to index both direct and indirect keyword occurrences. While this causes considerable redundancy in the content-related part of the index structure, it improves its retrieval performance especially for queries with unselective paths (see chapter 5 for sample queries and a performance analysis). One may expect that the indexing of indirect keyword occurrences is most effective for the ordinary DataGuide which, unlike the CADG in either realization, has to search an index subtree without any clue as to where the query keywords actually occur. However, the benefit of content-aware navigation is supposedly limited to path-selective keywords. When querying keywords whose occurrences are referenced by virtually every node in the index tree, the CADGs are forced to search most paths, just like the ordinary DataGuide. In these cases, indexing indirect keyword occurrences enhances query evaluation for all tested index structures—provided the textual query nodes are reached by soft edges.

Data structures. The CADG's only data structure affected by this optimization is the content/annotation table. (The following discussion applies analogously to the ordinary DataGuide.) It needs to hold an additional set $D^i_{k,i}$ of document node IDs for each index node/keyword pair (i, k), listing all document nodes referenced by i which *indirectly* contain an occurrence of k. The existing set $D^d_{k,i}$ of *direct* occurrences remains unchanged. Figure 3.10 shows the content/annotation table from figure 3.1 *(b)* on page 27 with indirect occurrences added. The table has been augmented by several subcolumns, such that each index node has subcolumns for all keywords referenced anywhere in its subtree. For instance, the index node #4, which only had a single subcolumn for the empty word ε before, now also subsumes the keywords *"index"* and *"XML"*, which occur in document nodes referenced by its descendants #5 and #6. The two new fields in #4's third row are left blank, which indicates that #4 itself lacks occurrences of these two keywords. Those fields which existed before are untouched. However, a fourth row has been added for the sets $D^i_{k,i}$. For instance, #4's fourth row reveals that its two documents nodes, &4 and &7, both have descendants where the keyword *"index"* occurs. These are the nodes &5 and &8 which are referenced by #4's child #5. By contrast, the keyword *"XML"* only occurs in &8.

[22] Let N be the set of document nodes and $N_c \subset N \times N$ the set of parent/child pairs in the document collection. Then the (irreflexive) ancestor/descendant relation, or parent/child closure, on document nodes is the set $N \times N \supset N_d = \{(n_u, n_v) \in N \times N \mid [(n_u, n_v) \in N_c] \lor [\exists n_w \in N \mid (n_u, n_w) \in N_d \land (n_w, n_v) \in N_c]\}$.

This is why the *"XML"* subcolumn for #4 contains a single document node ID, which belongs to &8's parent &7.

#0				#1			#2		#3			#4			#5		#6
ε	"index"	"survey"	"XML"	ε	"index"	"survey"	"index"	"survey"	ε	"index"	"XML"	ε	"index"	"XML"	"index"	"XML"	"index"
&0				&1			&2	&2	&3			&4;&7			&5;&8	&8	&6
	&0	&0	&0		&1	&1				&3	&3		&4;&7	&7			

Figure 3.10: Structure-centric content/annotation table with indirect keyword occurrences

Indexing algorithm The sets $D^i_{k,i}$ appearing in the fourth row of the content/annotation table in figure 3.10 are computed from the existing sets $D^d_{k,i}$ at indexing time. To this end, the indexing procedure is modified as follows. Assume an occurrence of the keyword $k \neq \varepsilon$ in the document node d is being indexed for the index node i. In any case, d is added to the set $D^d_{k,i}$ of direct keyword occurrences in i's subcolumn for k. If indirect keyword occurrences are indexed, then all sets D^i_{k,i_u} for any ancestor i_u of i need to be updated. Let d_u denote the ancestor of d which is referenced by i's ancestor i_u. Then d_u is added to D^i_{k,i_u}.

For example, during the creation of the CADG shown in figure 3.3 on page 32 the occurrence of the keyword *"XML"* in the document node &8 was indexed by the index node #5. First of all, &8 became the first element of #5's set of direct occurrences in its *"XML"* subcolumn. To produce the content/annotation table in figure 3.10 above, however, the indexing algorithm also propagated the newly added occurrence up the path from #5 to the root of the index tree. The three ancestors of #5 on this path, #4, #3 and #0, all had their sets of indirect occurrences updated. In the last row of #4's *"XML"* subcolumn, &8's parent &7 was added. The same happened to &7's parent &3 in the *"XML"* subcolumn of #4's parent #3, and finally to &0 and #0 (refer to figure 2.2 *(b)* on page 17 for the underlying document collection). In all three propagation steps, a new subcolumn for the keyword *"XML"* was added to the respective index node's column in the content/annotation table because it was the first (and only) time that an occurrence of this keyword was being indexed.

Retrieval algorithm The retrieval algorithm of an index structure supporting indirect keyword occurrences is very similar to the procedures already described in sections 2.4.3 and 3.2.2.2. Only soft-edged textual query nodes are handled differently. It has already been mentioned that indexing indirect keyword occurrences avoids an exhaustive search in that part of the index tree where document nodes with the right label path prefix are referenced. Consider e.g. the query `/book[./preface/paragraph["index"] and ./*[.//"XML"]]`, which has been introduced in section 2.4.3 for the DataGuide's retrieval algorithm. Its graphical representation is given in figure 2.1 *(b)* on page 13. The left branch of the query tree is processed as described for index structures without support for indirect keyword occurrences (see sections 2.4.3, 3.2.3.1, and 3.2.3.2). The right branch ends in a textual query node, $5, which is linked to its parent $4 by a soft edge. Recall from the aforementioned sections that $4 is matched by the index node #3. Instead of accessing all descendants of #3 one by one to fetch their respective occurrences for the keyword *"XML"*, whose ancestors referenced by #3 are then retrieved, an optimized index simply looks up the matches in #3's set of indirect occurrences of the keyword *"XML"* (see the last row in figure 3.10 above). In both cases, &3 turns out to be the only document node matching the query node $4. But the optimized procedure needs a single database access to compute the result instead of four (one for each node in #3's subtree). Besides, it does not perform any set operations on document node IDs.

3.3.2 Structural node identification of document nodes

Overview. Several techniques have been proposed in the literature to encode structural information in node IDs. A short overview is given in [Wei02], where the term *structural node identification* is used collectively for these techniques. The approaches reviewed there are all restricted to tree structures, being based either on a depth-first or a breadth-first traversal. Depending on the employed identification technique, certain structure tests can be performed on node IDs without accessing the tree itself. The supported operations range from ancestor/descendant tests to the computation of parent and child IDs for a given document node. Applied to the document tree, structural node identification can enhance query evaluation, especially phase 4 (see the discussion of the retrieval algorithm below). Apart from the DataGuide and CADG, it is compatible with many other navigational index structures, as discussed in chapter 6—provided there are no external constraints on node IDs in the document collection which prohibit the use of such techniques.

In this work, a depth-first based identification scheme proposed in [LM01], called *interval encoding* in [Wei02], has been applied to document node IDs and tested with all of the above index structures. The basic idea of interval encoding is to enable ID-based ancestor/descendant tests in a tree by attaching to each node's ID n the size t_n of the subtree this node spans (or, to be more precise, the number of its irreflexive descendants). The set of (reflexive) descendants of any node then comprises all nodes whose ID lies in the interval (n, t_n), i.e. whose ID is equal to or greater than the alleged ancestor's ID without exceeding the upper bound of its interval. For instance, consider document node &3 in figure 3.11, which shows the document collection from figure 2.2 *(a)* on page 17 with interval-encoded node IDs. The subtree rooted at &3 comprises the nodes &4 to &8. Therefore $t_{\&3} = 5$. Any descendant of &3 can be recognized from its ID, which lies between 3 and $3 + 5 = 8$, inclusively. As expected, this is only the case for the nodes &4 to &8, plus &3 itself.

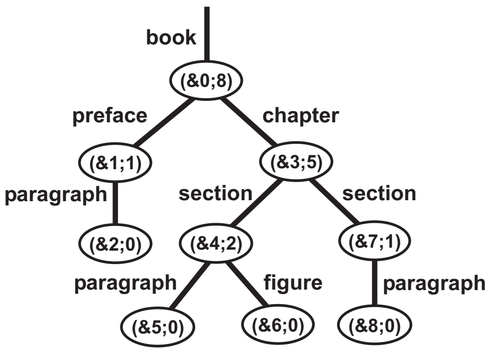


Figure 3.11: Interval encoding for a document tree

Data structures. One way to adapt a DataGuide or CADG to interval encoding is to use node ID pairs instead of singleton values. Each ID pair consists of an actual ID and the corresponding interval size, as described above. Alternatively, secondary indices mapping an ordinary node ID to its interval size could be used.

Indexing algorithm. The intervals of all document nodes are computed at indexing time. Depending on the parsing strategy, this may, but need not entail a second pass through the documents being indexed. In case the document collection is indexed in a depth-first traversal (as by common XML parsers) and interval encoding is used, creating intervals is trivial and does not substantially alter the indexing algorithm. Nevertheless, more effort may be required, e.g. when applying a breadth-first identification scheme in a depth-first traversal of the documents.

Retrieval algorithm. As mentioned above, the use of interval-encoded document node IDs mainly affects retrieval phase 4, where two paths of a query tree are joined. In the DataGuide's retrieval algorithm described in section 2.4.3, which applies analogously to the CADG in either realization, this involves an ancestor/descendant test for the retrieved document nodes. Without optimization, the test usually requires at least one database look-up, whereas with interval encoding it can be performed in constant time as a main-memory operation.

3.3.3 Structural node identification of index nodes

Overview. In much the same way as with document nodes, structural node identification can be applied to index nodes, namely to enhance navigation (retrieval phase 1). Especially the ID-Comparison CADG can be expected to profit from this optimization, its relevance checks relying on index node ancestor/descendant tests. But also the Signature CADG's and the ordinary DataGuide's navigation can be enhanced. In conjunction with a label index (see the next subsection), interval encoding on index nodes sometimes saves an exhaustive search in large parts of the navigational structure. Although regarding only main-memory operations, these optimizations still promise a performance gain for the huge index trees resulting from highly diverse and complex document collections.

Data structures. Analogously to document nodes, interval-encoded index nodes need to be either supplied with tuple IDs or used with a separate interval index. Figure 3.12 depicts the navigational structure of the CADG in figure 3.3 *(a)* with interval-encoded index nodes.

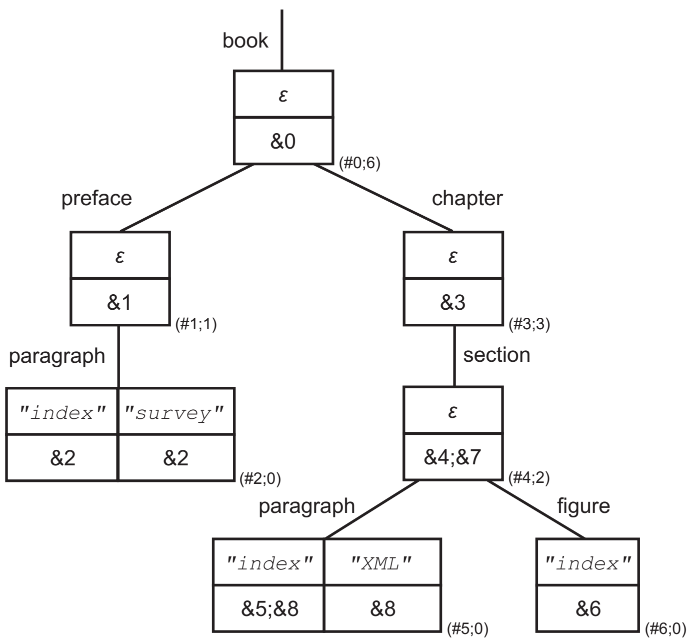


Figure 3.12: Interval encoding for an index tree

Indexing algorithm. An extra postprocessing of the index tree is needed to compute intervals on index nodes. The reason why this cannot be integrated with index creation is that additional nodes may be created at any time while new document paths are being explored, changing the size of their parents' subtrees. This hints at a major drawback of structural node identification, which is its lack of flexibility with respect to updates of the underlying data structure. If e.g. new documents are added to a collection whose indices are optimized with interval encoding, a complete reindexing is necessary under certain conditions. Notwithstanding, there are techniques to avoid this in many cases, as surveyed in [Wei02] and elsewhere.

Retrieval algorithm. Here only the navigation relevance check of the ID-Comparison CADG is considered; for an application of interval encoding to the other indices, refer to the next subsection.

As described in section 3.2.3.1, the ID-Comparison CADG's navigation relevance check involves ancestor/descendant tests on index nodes. When matching a query path ending in a textual query node, each index node reached during navigation needs to be compared to the set of index nodes referencing relevant document nodes. Depending on the query keywords' path selectivity as well as on the size of the index, there may be many such nodes, which explains that under certain circumstances the overhead for query preprocessing leaves the ID-Comparison CADG outperformed by other, e.g. heuristic indices (see chapter 5 for examples of such cases). Structural node identification of index nodes can attenuate this effect, allowing the ancestor/descendant test to be performed in constant time and without any navigation in the index. Yet its benefit is limited compared to the performance gain achieved for document nodes, because there expensive database operations are avoided, whereas in the case of index nodes only main-memory operations are optimized, as pointed out above.

3.3.4 Label index for enhanced index navigation

Overview. A secondary *label index* is another means to enhance retrieval phase 1. It has been specifically designed for query paths dominated by labelled soft edges to structural query nodes. Recall from section 2.4.3 that evaluating soft-edged structural query nodes results in an exhaustive search of the navigational structure, starting from the last matching index node. If the soft edge is labelled, only those nodes in the subtree being searched can enter retrieval phase 2 whose incoming edge has the right label. As an extreme case, consider a label which does not exist in the entire index tree. Evaluating a query path where such a label is attached to a soft edge involves a possibly expensive search of the navigational structure, which is doomed to fail from the very beginning. A simple yet effective remedy is a secondary index mapping each label occurring in the document collection to the set of index nodes reached by an edge with that label. In the aforementioned case, a single look-up in the label index would immediately prove the query to be unsatisfiable, thus saving the whole navigation. On the other hand, using a label index may admittedly slow down query evaluation when trying to match a very frequent label in a small subtree, e.g. on the next-to-last level in the index tree. Very likely there are effective heuristic methods to determine when the use of a label index is worth while, which could be applied either at indexing or at query time. In this work, however, only a proof of concept was intended. Hence the label index has been used either for all queries collectively or for none at all, independent of the characteristics of individual queries or the underlying document collection.

Data structures. The label index is a simple mapping $lab : l \mapsto I_l$ from labels to sets of index nodes. For each label l in the label index, I_l contains exactly those index nodes which are reached by an edge labelled l. For most document collections, except those whose underlying schema is extremely complex, the index can be supposed to fit main memory. While it could alternatively be stored on disk, this would probably disqualify it as an optimization of a main-memory operation like navigating index paths.

Figure 3.13 shows the label index for the CADG in figure 3.3 on page 32.

book	chapter	figure	paragraph	preface	section
#0	#3	#6	#2; #5	#1	#4

Figure 3.13: Label index

Indexing algorithm. At indexing time, whenever a new index node with an incoming l edge is created it is added to the set I_l in the label index.

Retrieval algorithm. As pointed out above, the label index is used only when labelled soft edges are evaluated. Let i be an index node matching the parent of a query node q_s whose incoming soft edge has the label l. Instead of searching for q_s's matches in the index subtree rooted at i, all descendants of i in the set $I_l := lab(l)$ fetched from the label index are collected for further path matching. (Note that this ancestor/descendant test may be enhanced by interval encoding in index nodes, as described in the previous subsection.) The resulting subset $I_{l,i} \subset I_l$ contains exactly those nodes in the subtree which are reached by an incoming l edge, hence the modified retrieval algorithm yields correct results.

For example, consider query `/book/chapter//paragraph` being evaluated with the CADG in figure 3.3 and its label index, shown in figure 3.13 above. The query path prefix `/book/chapter` leads directly to the index node #3, whose subtree (including #4, #5 and #6) would be searched for paragraph edges if no label index were provided. Instead the two members of $lab(l) = \{\#2, \#5\}$ are examined to identify descendants of #3 among them. While #2 is discarded, #5 passes the test. In this case, the look-up in the label index involves two ancestor/descendant tests, as opposed to accessing three nodes in #3's subtree—a fairly balanced effort. If #3's subtree were much bigger though, or conversely the set $lab(l)$, costs and benefits of either solution would be easier to contrast.

Chapter 4

Implementation

This chapter summarizes the programming accomplished in the course of this work. The two realizations of the Content-Aware DataGuide presented in section 3.2.3, ID-Comparison CADG and Signature CADG, have been implemented and integrated with an existing retrieval system, the CIS System. As explained in section 4.2.3, two different ways to handle keyword signatures have lead to a further specialization of the Signature CADG into *Signature Look-Up CADG* and *Signature Generation CADG*. Besides, the DataGuide has been implemented for comparative evaluation with the CADG, such that in the end four different index types have undergone the performance tests discussed in the next chapter. Apart from the index structures proper, various adaptions of the CIS System were necessary to create a suitable testbed for the experiments.

First, the original CIS System (i.e. without the new index structures) is reviewed. Then the newly implemented packages are presented, along with some observations regarding their overall design as well as selected data structures and operations. The following section deals with the integration of new index structures into the system. Finally, adaptions of the original query evaluation algorithm to the new index structures are mentioned briefly.

4.1 Testbed: the CIS System

The *CIS System* [Meu00] was developed at the Centre for Information and Language Processing (CIS), University of Munich, Germany. It serves as a testbed for evaluating new retrieval techniques for tree-shaped Semi-Structured Data. The original system features a query formalism similar to the one described in section 2.3, a single index structure, the CIS Index (see section 4.1.2), Complete Answer Aggregates (CAAs) as a datastructure for query results (see section 4.1.3), and a simple result ranking mechanism. In this section, a brief overview of the system in its original state is given. The implementation accomplished in the course of this work is described in section 4.2, which also comments selected modifications of the original system.

4.1.1 Module overview

Since the CIS System was created as an infrastructure for testing Complete Answer Aggregates (see section 4.1.3), the root package is called `caa`. The core of its functionality resides in the two classes `caa.index.CISIndex` and `caa.datastructure.agg.Aggregate`. The former implements the tree index CIS Index (see section 4.1.2). Additionally, it triggers document parsing, index creation, and retrieval. The latter implements the CAA data structure, and also contains the query evaluation algorithm. A third component is the graphical user interface to the CIS System [Sch02]. As described in section 4.1.2, the CIS Index maintains a path table, which is stored in a relational database back-end. In the case of the CIS System, the PostgreSQL database system [PSQ] is used. It is accessed via the JDBC interface [JDB] provided with the Java libraries.

4.1.2 CIS Index

The index structure used so far with the CIS System is the *CIS Index* [Meu00], a path look-up index for tree-shaped document collections. Its only data structure is a *path table* mapping a given keyword to the label paths of all document nodes where this keyword occurs. Each label path is annotated with the IDs of its nodes, from the root down to the document node containing the indexed keyword occurrence. For instance, in a CIS Index created for the document collection shown in figure 2.2 *(b)* on page 17, there is an entry mapping the keyword *"XML"* to the label/ID path /book$_{\&0}$/chapter$_{\&3}$/section$_{\&7}$/paragraph$_{\&8}$. Keywords occurring in more than one document node, like *"index"*, are mapped to multiple paths.

Processing queries with the CIS Index is substantially different from the procedures described for the DataGuide (see section 2.4.3) or CADG (see sections 3.2.2.2, 3.2.3.1, and 3.2.3.2). The reason is that the CIS Index treats document paths as atomic units rather than compositional objects, which precludes navigation. Furthermore, the content/structure join is realized implicitly by the structure of the path table, which saves an explicit join phase as required for the DataGuide. As a consequence, the retrieval algorithm for the CIS Index goes through only two phases: first, label/IDs paths are fetched for all query keywords, which are then joined in case of a tree query. The second step is obsolete when processing path queries. A path joining algorithm related to the construction of Complete Answer Aggregates (see the next section) is described in [Meu00].

The CIS Index has two advantages: it is simple to implement, fitting any relational database system, and it avoids the path reconstruction problem (see section 5.4.4) by storing annotated document paths in their integrity. However, this feature is also responsible for its main drawback, which is redundancy. As pointed out by [Wei02], whenever a document node has more than one child node with textual content, its path is duplicated in the index. The reason is that each keyword occurrence in any of its child nodes is stored as a separate entry in the CIS Index table. Since such an entry contains the complete path from the document collection's root to the node where the keyword occurs, the path prefix leading to the parent node is stored once for each keyword occurring in one of the child nodes. Not only is this an obstacle to incremental index updates; it also causes a considerable storage overhead compared to navigational index structures, such as the DataGuide or CADG. See section 6.5 for a short review of the CIS Index and its trade-off between storage and retrieval performance.

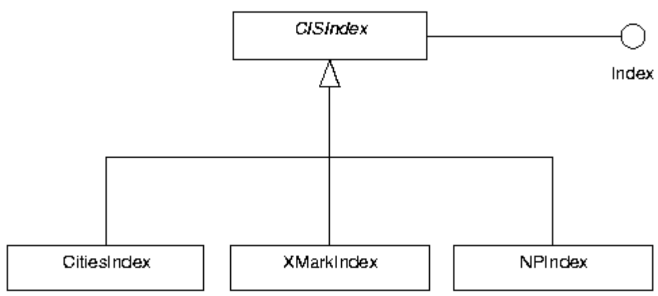


Figure 4.1: The CIS System's index hierarchy in `caa.index`

Figure 4.1 shows the original CIS System's index class hierarchy, which resides in the package `caa.index`. The core index functionality as specified in the interface `Index` is part of the abstract class `CISIndex`, which is extended by three document-collection specific subclasses corresponding to the three document collections described in section 5.2.2. All three subclasses realize the CIS Index slightly differently to optimize the path table layout for a specific document collection. As a consequence, they must implement the methods for path table creation and retrieval, which are left abstract in `CISIndex`.

4.1.3 Complete Answer Aggregates

A *Complete Answer Aggregate* [Meu00, MS01, Meu98], or CAA for short, is a concise representation of the results retrieved for a tree query. Being itself tree-shaped, its overall structure reflects the structure of the query tree, i.e. each node (or *slot*) in the aggregate tree corresponds to exactly one structural query node. Furthermore, the aggregate nodes contain references to all document nodes matching the corresponding query node. In a way, this feature resembles the index annotations described for the DataGuide in section 2.4.1. Note, however, that the CAA's references to document nodes keep track of hierarchical relationships between individual document nodes, unlike index annotations. In other words, when representing the results of a given query as a Complete Answer Aggregate, no information about the matching document paths is lost, although the CAA makes it much easier to browse the hits from the document collection. Yet the main advantage of CAAs as a result data structure is the fact that both the time needed to create them and the space they occupy is polynomial and sometimes even linear in the size of the document collection, even though the number of answers represented by the aggregate may be exponential.

Figure 3.7 on page 40 depicts a CAA for the query from figure 2.1 *(a)* (see page 13). In this simple example, only one document node is referenced by each aggregate slot. Each parent/child pair of document nodes is linked such that the original document paths from figure 2.2 *(b)*, /book$_{\&0}$/preface$_{\&1}$/paragraph$_{\&2}$ for the left branch and /book$_{\&0}$/chapter$_{\&3}$/section$_{\&7}$/paragraph$_{\&8}$ for the right, are restored in the aggregate.

4.2 Implementation details of the CADG

This section presents the implementation of the CADG and related parts of the retrieval system as it has been accomplished in the course of this work. First a concise overview of the class hierarchy is given. Then selected parts of it are commented in more detail to highlight some of the project's design rationales. Finally, a few interesting data structures and operations are examined.

4.2.1 Package structure

4.2.1.1 Overview

The code written for this work amounts to a total of 50,000 lines, including about 20,000 comment lines (both in-line and Javadoc, plus CVS log). The code is distributed over seven subpackages of `caa`, three of which have existed before. The CIS System's query and Complete Answer Aggregate classes (packages `caa.datastructure.query` and `caa.datastructure.agg`, respectively) have been reused to a large extent. In the following, the most important among the newly implemented classes and packages are presented, each with a UML class diagram and a short description. Utility classes, such as exceptions and toolboxes, have been omitted.

4.2.1.2 Package `caa.db`

The package `caa.db` contains the backbone classes for the new implementation of the retrieval system. Its core functionality (including the query evaluation algorithm) is provided by the `DBManager` class (see section 4.2.2.3), which is used by the command-line interface `DBInterpreter` (see section 4.2.2.4). The shaded classes in the right half of the diagram are not needed for retrieval, but for analyzing and formatting performance results obtained during retrieval. They are also interfaced by `DBInterpreter`, and access the database back-end via `DBManager`. Note that `DBManager` is never used directly, but always by means of `DBInterface`. This facilitates the reuse of the other classes with a different retrieval system manager.

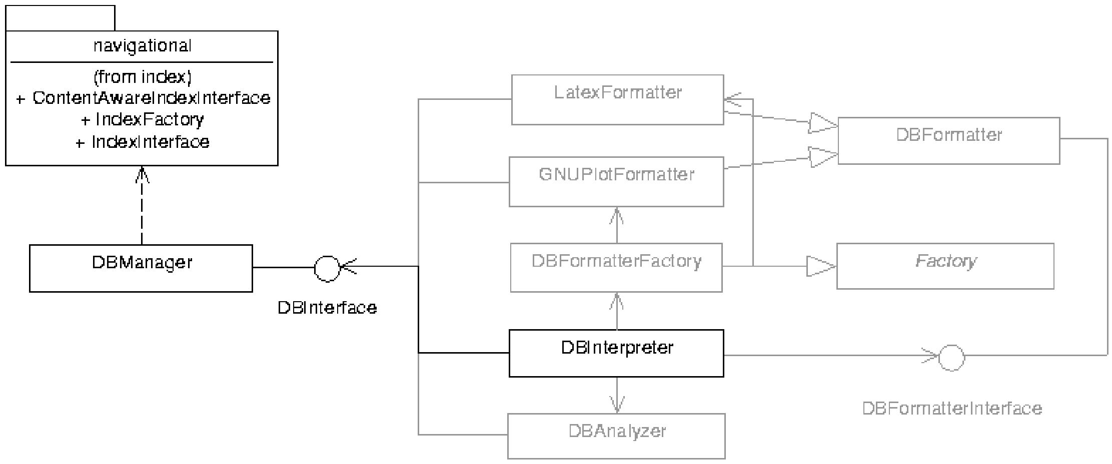


Figure 4.2: The new retrieval system package `caa.db`

4.2.1.3 Package `caa.index.navigational`

The `navigational` subpackage of `caa.index` has been created to separate the implementation of the CADG and DataGuide from the existing CIS Index. It contains the three index classes `DataGuide`, `IDComparisonCADG`, and `SignatureCADG`, in order of increasing specialization. As shown in the left part of figure 4.3, their hierarchy is paralleled by the three interfaces `IndexInterface`, `ContentAwareIndexInterface`, and `SignatureIndexInterface`, although only the first two are currently used by `caa.db.DBManager`. The different index classes can be instantiated with the help of the `IndexFactory` class, which extends an abstract factory specification class `caa.datastructure.generic.Factory`. Other datastructures imported from the new `caa.datastructure.generic` package (see the next subsection) include node IDs and signatures for the `IndexNode` and `SignatureIndexNode` classes (shaded in figure 4.3).

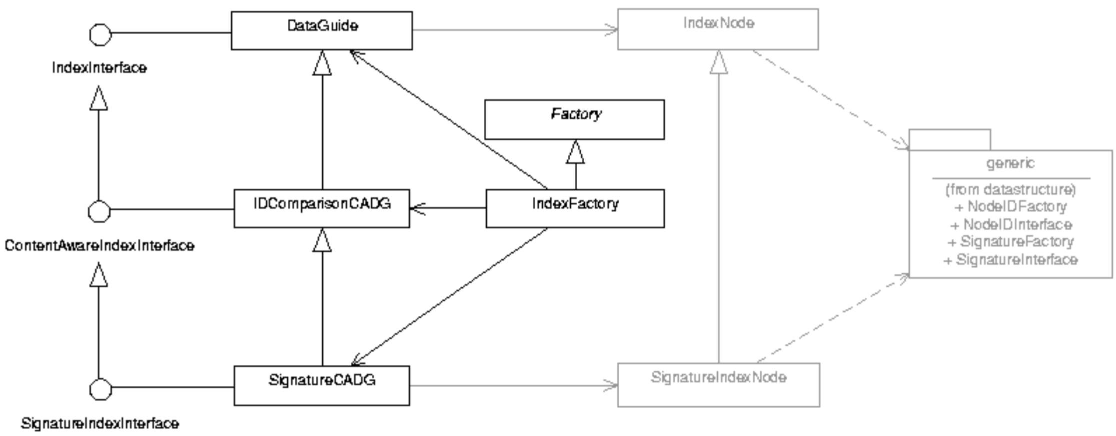


Figure 4.3: The CADG index hierarchy in `caa.index.navigational`

4.2.1.4 Package `caa.datastructures.generic`

The new subpackage `generic` in the `caa.datastructure` package contains reusable datastructures which are neither related to queries (in `caa.datastructure.query`) nor to Complete Answer Aggregates (in caa.datastructure.agg). These are on the one hand various kinds of node identifier and on the other hand signatures of different sizes. Figure 4.4 depicts the hierarchy of node ID classes and interfaces. Currently only two types of IDs for depth-first numbering are used in the system: plain integer IDs (implemented by the DepthPhysIntNodeID class) and interval-encoded IDs (implemented by the DepthIntervalIntNodeID class). To prepare possible extensions, five more classes are part of the hierarchy (shaded in figure 4.4). For instance, BreadthVirtIntNodeID shall implement IDs to be used with the Virtual Nodes numbering scheme [LYYB96]. All classes in the hierarchy extend IntNodeID as the most general specification of a numeric node identifier. They are instantiated by the class NodeIDFactory, which specializes an abstract factory class Factory (see section 4.2.2.1 for a brief discussion of the factory design pattern). Factory provides datastructures and operations for handling and storing factory properties, which determine which of the registered classes to instantiate when the method Factory.getTypedNewInstance() is called.

As figure 4.4 shows, there are three interfaces associated with the hierarchy of node ID classes. The most general one, NodeIDInterface, declares methods for comparing, copying, and manipulating the values of node IDs. HierCompNodeIDInterface extends NodeIDInterface by further comparison functionality for testing hierarchical relationships. Its two methods most relevant to caa.db.DBManager are isAncestorOf() and isDescendantOf(). A third interface, UpNavNodeIDInterface, which is currently unused, provides additional methods for in-memory path reconstruction. This feature seems most promising for future performance enhancement, as discussed in section 5.4.4.

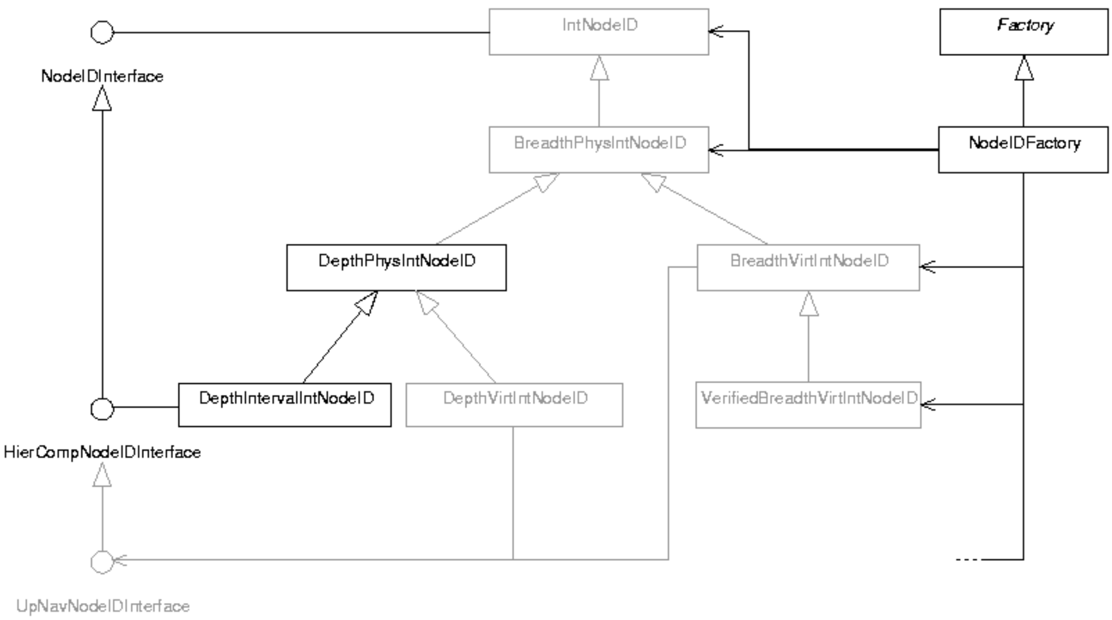


Figure 4.4: Node ID classes in caa.datastructure.generic

The package caa.datastructure.generic also contains several classes related to signatures. Most prominently, they are used by the caa.index.navigational.SignatureCADG class to represent navigation and occurrence-fetching signatures (see section 3.2.3.2), as well as signatures attached to the nodes in the query tree. But the classes are sufficiently generic to support any other use of signatures. For instance, caa.db.DBAnalyzer and caa.db.DBFormatter both use the classes from this package to represent index configurations and query classes, two concepts related to the performance tests suites described in chapter 5.

As figure 4.5 illustrates, there are four signature classes in the package, all of which implement the interface `SignatureInterface`. Each class implements signatures of a fixed size, as their names suggest. The class used for retrieval purposes throughout experimental evaluation, `LongSignature`, provides 64-bit signatures. The formatter and analyzer classes from `caa.db` use `ByteSignature` objects instead, which have only 8 bits. Signatures are created with the help of a factory class, similar to node IDs (see above).

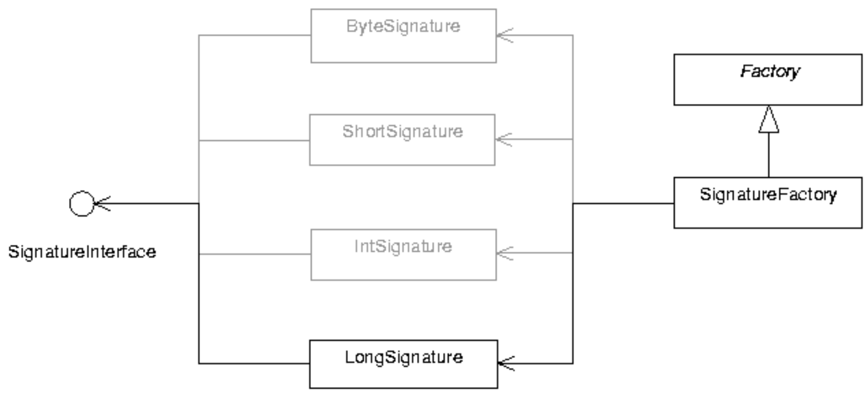


Figure 4.5: Signature classes in `caa.datastructure.generic`

4.2.2 Design issues

4.2.2.1 Inheritance and polymorphism

As the class diagrams in section 4.2.1 prove, the new implementation of the retrieval system makes extensive use of inheritance and polymorphism as the two main characteristics of the object-oriented paradigm. Typical examples where common functionality is shared via inheritance include the hierarchy of index classes in figure 4.3 and the node ID hierarchy in figure 4.4. In the former, the `SignatureCADG` class inherits all retrieval methods from its superclass `IDComparisonCADG`. Both classes also share the same index creation functionality, which `SignatureCADG` augments with signature handling. `IDComparisonCADG` (and consequently `SignatureCADG`) inherits some fields from the `DataGuide` class, which hold datastructures like e.g. label index or a node ID factory (see the next subsection). In the case of the node ID classes, common functionality like comparison, copying or incrementing and decrementing is shared via inheritance.

The two examples also demonstrate the use of polymorphism throughout the new implementation. In both cases there are multiple variants of the same logical component, which is represented by an interface (`caa.index.navigational.IndexInterface` for the three different index structures, and `caa.datastructure.generic.NodeIDInterface` for the node IDs). The same applies to the signature classes and their interface `caa.datastructure.generic.SignatureInterface` (see figure 4.5), as well as to the different subclasses of `caa.db.DBFormatter` (see figure 4.2). [Coo98] suggests the *Factory pattern* when runtime information is used to create instances of different classes which are either derived from the same abstract superclass, or implement a common interface. In the aforementioned examples, there is a dedicated factory class for each interface which allows to instantiate any of the implementing classes based on the parameters it is given at runtime. The actual parameter values are supplied by the calling method in `caa.db.DBInterpreter`. For instance, the interpreter's `createIndex` command takes several parameters describing which type of index to create with what kind of optimizations (see chapter 3). The command ultimately results in a call to the `createIndex` method in `DBManager`, which in turn uses an instance of the `IndexFactory` class to create the right kind of index object based on the type parameter it received from `DBManager`.

4.2.2.2 Generic document parsing and indexing

For historical reasons, both the document parser and the index classes in the original CIS System (see section 4.1) are specific to the underlying document collection. This means that whenever new data are indexed, the index and parser classes used so far need to be adapted or replaced. The new implementation solves this problem by providing generic parser and index classes (in the `caa.parser.generic` and `caa.index.navigational` packages, respectively) suitable for any document collection (see section 4.2.1.3).

4.2.2.3 Central retrieval system manager

A salient deficiency in the original CIS System's design is related to the role of the `CISIndex` class. Although conceived as an implementation of the CIS Index (see section 4.1.2), it also serves as a sort of manager class triggering document parsing, indexing, and retrieval. This is clearly more than what an index is expected to do. Another peculiarity in the design of the CIS System is that its query evaluation algorithm is implemented by the class `caa.datastructure.agg.Aggregate`, i.e. considered a part of the Complete Answer Aggregate to be produced as query result. What seems to be missing is a central management unit for the retrieval system, with a variety of operations reflecting all aspects of the system's functionality. In the new implementation, there is a class `caa.db.DBManager` charged with this task (see section 4.2.1.2 above). It implements the interface `caa.db.DBInterface` as an abstract description of the operations supported by the retrieval system. `DBInterface` represents the system's back-end which may be accessed by different and completely independent front-ends. As mentioned above, there exists a graphical user interface (GUI) for the CIS System. Additionally, a command line front-end with scripting support has been implemented for evaluation purposes (see the next subsection).

4.2.2.4 Command-line interpreter

As mentioned above, a new command-line based front-end has been added to the system, serving as a central interface to all facilities provided by the retrieval system. The front-end class `caa.db.DBInterpreter` currently supports two dozen commands for document handling, index creation and maintenance, retrieval, performance analysis, formatting of performance results, and query generation. They are listed in appendix C, along with their respective parameters. Once interpreted, each command triggers an appropriate method in `caa.db.DBManager`, `caa.db.DBAnalyzer`, one of `caa.db.DBFormatter`'s subclasses, or `caa.query.QueryGenerator`. The interpreter accepts input either interactively, which is valuable for productive use of the system, or in any number of batch files which can also be nested. `DBInterpreter`'s batch mode is essential for extensive test suites as the ones described in chapter 5.

Interpreters are a common approach to providing complex functionality in a coherent and centralized manner. [Coo98] recommends the *Interpreter pattern* for programs supporting different kinds of output. As shown in figure 4.2 on page 54, the `DBInterpreter` currently formats performance analysis data in LaTeX and GNUPlot format. Another reason why the Interpreter pattern is favourable for the system's command-line interface is that unlike a GUI, it allows to combine interactive and batch processing. For test suites with automatically generated queries such as the ones carried out in the course of this work, the batch files serve three purposes simultaneously. First, they constitute the interpreter's input. Second, the batch file commands are logged in the output files produced by the interpreter, which allows to review a session, and in particular to monitor the individual queries' retrieval results. The output files may be passed to the analysis tool `caa.db.DBAnalyzer`, which relies on the logged query commands to associate queries from the batch files with their respective results. Lastly, the commands in the output files can serve as a basis for creating modified query sets with the help of the `caa.datastructure.query.QueryFileGenerator` class. Figure 4.6 illustrates this "document workflow" among the different tools.

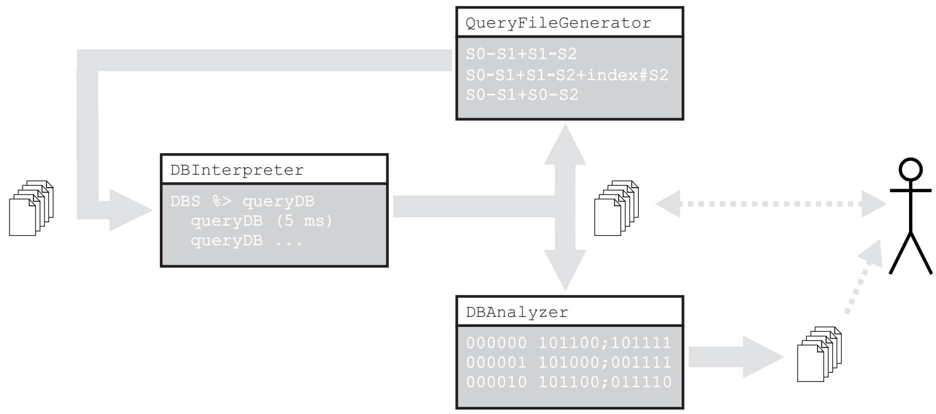


Figure 4.6: "Document workflow" among the tools in `caa.db` and `caa.datastructure.query`

4.2.2.5 Database connection pool

Among other tasks, the `DBManager` class is responsible for providing access to the database back-end. As in the original CIS System, database connectivity is realized using the JDBC API [JDB]. The PostgreSQL system used as back-end starts a new server process for each `java.sql.Connection` opened. When the `caa.query.QueryGenerator` class was developed, it turned out that running numerous database servers in parallel may tie up a considerable part of the resources available, especially when they are all busy processing queries to be classified. To minimize the number of database server processes, and to gain central control over the corresponding `Connection` instances, the `DBManager` class maintains a *connection pool* as described in [PDP02].[23] Any class may request an open `Connection` to the current database by calling `DBManager.getConnection()`. Beforehand a running database server needs to be addressed using `DBManager`'s `connect()` method. The `getConnection()` method returns already opened connections which are currently idle, if possible. Only when no such object exists, a new `Connection` instance is created (which causes a new database server process to be launched). Connections which are not used any more can be put back into the pool by calling `DBManager.releaseConnection()`.

4.2.3 Data structures and operations

Section 3.2.3.2 mentions that keyword signatures for the Signature CADG are either stored in the database and looked up when needed, or created on the fly at query time. The first possibility, which turns the Signature CADG into a *Signature Look-Up CADG*, or SL for short, involves an additional database access during query preprocessing.[24] Besides, a *signature table* is necessary to store a signature for each of the indexed keywords. Experiments show that the additional disk space occupied by the signature table may amount to roughly 20% of the index size when the index is large, or even 50% and more for small indices (see table 5.9 on page 81, where SL is compared to the *Signature Generation CADG*, abbreviated SG). Creating signatures on the fly, as the Signature Generation CADG does, saves not only the signature table, but also the signature creation step in retrieval phase 0. On the other hand, creating signatures for many query keywords may take longer than fetching them from the database, depending on the complexity of the creation procedure (see section 5.4.2.2 for empirical observations on this topic). In this work,

[23] The PostgreSQL driver's connection pooling facilities, implementing new features introduced with the JDBC 2.0 Standard Extension, could not be used since as of PostgreSQL 7.3, there is no way to close an open connection (see [PDP02] for details).

[24] The signatures for all query keywords are fetched at once to avoid unnessary disk operations.

the following signature creation algorithm has been implemented for the Signature CADG (with $numSigBits := 64, numSetBits := 3$):

```
// creates a keyword signature with a fixed number of bits set to 1,
// based on a hash code for the given keyword
procedure createSignature (Keyword k)

  // get the signature's size and number of bits to be set
  numSigBits := total number of bits in the signature to be created
  numSetBits := number of bits set to 1 in the signature to be created

  // prepare a signature to be returned
  s := bit string of length numSigBits, with all bits set to 0

  // determine the number of possible positions for every bit to be set
  // the signature is divided into this number of chunks each comprising numSetBits bits
  numChunks := numSigBits / numSetBits
  numChunkBits := number of bits needed to encode values between 0 and numChunks

  // get a hash code for the given keyword
  hashCode := integer hash code for k

  // the last few bits in the hash code are used to determine in which signature chunk a bit is to be set
  for b := 0; b < numSetBits; b := b + 1 do
    c := value encoded by the last numChunkBits bits of hashCode
    set the bth bit in s's cth signature chunk
    remove the last numChunkBits bits from hashCode
  end for

  // return the computed signature
  return s

end procedure
```

Algorithm 4.1: Signature creation with the Signature Content-Aware DataGuide

Another issue found relevant during the experimental evaluation in chapter 5 is the use of indices on tables stored in the back-end database. They are most valuable to speed up look-ups performed by any of the index structures presented in chapter 3. However, multi-dimensional indices, i.e. those built over complex keys, are slower than one-dimensional indices. While the DataGuide's content and annotation tables have only one-dimensional keys and indices (on keywords and index node IDs, respectively), the CADG's content/annotation table requires a two-dimensional index built over both keywords and index node IDs. The Signature Look-Up CADG's signature table behaves like the DataGuide's content table in this respect. A possible enhancement of the ID-Comparison CADG in this context is a separate look-up table for query preprocessing, analogous to the Signature Look-Up CADG's signature table. Recall from section 3.2.3.1 that in retrieval phase 0, the ID-Comparison CADG collects all index node IDs from its content/annotation table which reference document nodes where one of the query keywords occurs. Storing the sets of relevant index nodes in a look-up table not only avoids set manipulation at query time, but also allows to access a one-dimensional instead of a two-dimensional index during the preprocessing. This optimization has not yet been implemented.

4.3 Integration of the CADG with the CIS System

As mentioned in section 4.2.2.3, part of the management functions now shifted to the interface `caa.db.DBInterface` were originally fulfilled by the `caa.index.CISIndex` class, and therefore included in the interface `caa.index.Index` on which the CIS System's GUI relies. To prepare an integration of the new index structures within the GUI, the `DBManager` class from `caa.db` implements the `Index` interface from `caa.index`. This establishes a cross-package link between the former and the new retrieval system core, as shown in figure 4.7.

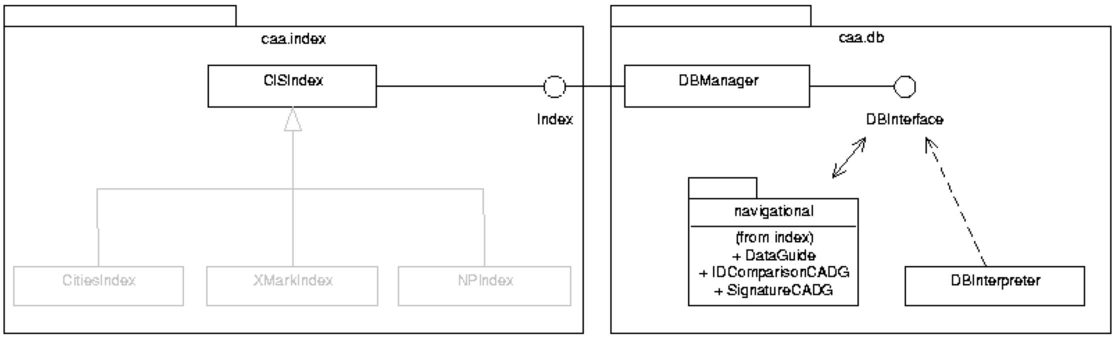


Figure 4.7: Integration of `caa.db` with `caa.index`

4.4 Query evaluation algorithm

As stated in section 4.2.1.2, queries are evaluated by the `caa.db.DBManager` class. The underlying algorithm, whose full listing and explanation exceeds the scope of this work, closely resembles the evaluation algorithm used in the original CIS System [Meu00]. In both cases, query processing and the creation of a Complete Answer Aggregate (see section 4.1.3) are intertwined. The same optimization techniques are applicable to both algorithms when matching label paths, filling slots with references, or joining multiple query paths. Particular attention has been paid to preserve the original algorithm's theoretical complexity of $\mathcal{O}(\, d \cdot log(d) \cdot h \cdot q\,)$ according to [Meu00], where d is the size of the document tree, h its height, and q the size of the query tree. To this end, e.g., the ID sets in all index tables are ordered, which also allows for binary instead of sequential search in annotations and keyword occurrences.

Differences between the old and the new query evaluation algorithm arise mostly from the respective index structure used. While the CIS Index resides entirely on disk and cannot be navigated, aggregate creation and index navigation are tightly coupled in the new algorithm. Since path reconstruction (see section 5.4.4) requires retrieved document nodes to be associated with their referencing index node anyway, the navigational structure is augmented by temporary data structures for query evaluation, such as candidate lists holding yet unconfirmed matches. Like the original algorithm, the new procedure avoids repeated path matching for the same label path prefix. For details on query processing in the context of CAAs, see [Meu00].

Chapter 5

Evaluation

In the previous chapters various index structures and optimizations have been presented and examined from a conceptual and implementational point of view. This chapter strives to assess their practical use based on empirical observations. To this end, all index structures and optimizations have been submitted to extensive and systematic experiments, whose complete results are given in appendix A. While here only selected issues are considered which seem most apt to contrast the strengths and weak-spots of the individual approaches, the reader is referred to the appended tables for further examination.

Before delving into the details of this performance evaluation, a brief overview of the CADG's retrieval performance and storage requirements compared to the DataGuide is given. It shows that the CADG is on average twice as fast as the DataGuide, and may under certain conditions outperform it by nearly two orders of magnitude. The following sections deal with the organization of the performance tests. First the various index configurations and test suites used for experimental evaluation are presented. Each test suite, which covers all index configurations, is characterized by the set of queries being evaluated and the document collection being queried. Queries are grouped into classes to prepare subsequent comparison and evaluation. Second, the testing environment is described shortly. Implementation details of the testbed can be found in section 4.1. The third part of this chapter is dedicated to the performance evaluation proper, which analyzes the results for all index types and optimizations presented in chapter 3. It is followed by a concise discussion of the storage requirements of the individual index structures. A final section concludes the evaluation with a few recommendations for the practical use of the different index structures and optimizations, based on the collected performance data.

5.1 Results in a nutshell

This section gives a concise overview of the performance results discussed in the remainder of this chapter. A more detailed description of the testing environment and organization follows directly hereafter. Figure 5.1 compares both the retrieval performance and storage requirements of the Signature Look-Up CADG and the DataGuide, each with the most effective optimizations applied. In the left plot, each pair of boxes represents the average retrieval time over all tested queries needed by either of the two index structures. There is a separate pair of boxes for each of the four test suites accomplished in the course of this evaluation. The individual test suites are discussed in section 5.2; in short, they stand for a particular set of queries being evaluated on a particular document collection. As one can see from the figure, the CADG performs 50% better than the DataGuide in all test suites, for the last one even more than 90% better. Once again, these data are obtained using the set of optimizations which makes both index types perform best on a given document collection.

The right plot in figure 5.1 illustrates the storage requirements of both Signature Look-Up CADG and DataGuide, relative to the size of the document collection being indexed. (The document collections are described in section 5.2.2.) The leftmost pair of boxes represents the smallest document collection, *Cities*, which takes up about 3 MB in the database. Here the storage overhead caused by the CADG is considerable, compared to the DataGuide which exceeds the original size of the data by 34%. In the case of the second document collection, *XMark*, the DataGuide is very compact, whereas the CADG still reaches nearly four times the size of the indexed data. The largest document collection, *NP* with more than 600 GB, is the most favorable for both index structures: the Signature Look-Up CADG's storage overhead is reduced to 20%, compared to 5% for the DataGuide. Obviously the storage overhead of both indices does not grow indefinitely with the size of the index data, but rather saturates for large document collections.

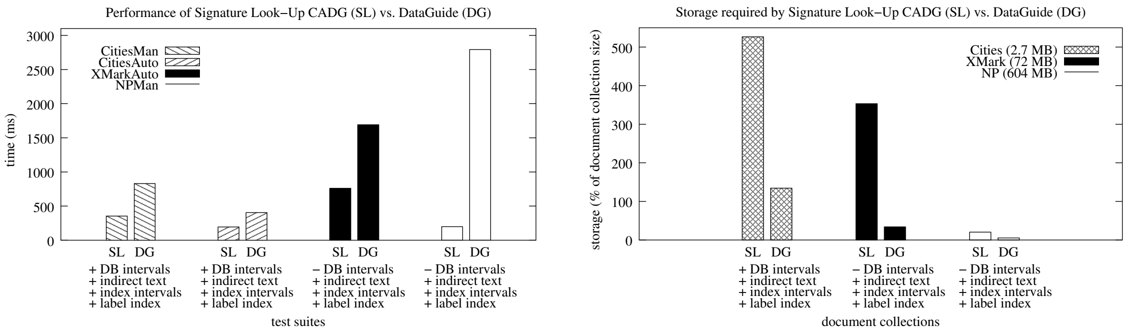


Figure 5.1: Time and space requirements of the CADG

These observations indicate that the CADG is a highly effective means to enhance retrieval performance, especially for huge document collections where its storage requirements are relatively modest compared to the indexed data. For smaller document collections, query evaluation is still significantly faster than with the ordinary DataGuide, but at the cost of a much higher storage overhead. Note, however, that in such cases the absolute size of the index is still manageable (about 10 MB for the *Cities* collection). More details about storage consumption are given in section 5.5.

5.2 Test suites

The experimental results presented here and in appendix A are organized as *test suites*. Each suite tests all *index configurations*, i.e. combinations of index structures and optimizations (see below), for a single document collection and a set of queries to be evaluated. In the course of this work, four test suites have been accomplished, each comprising between 12,000 and 67,000 individual query evaluations approximately. Apart from the absolute time interval needed for each evaluation, statistical aggregations (minimum, maximum, average, standard deviation, variance) have been computed. Besides, the results of all index configurations for a single query have been compared to the performance of a *reference configuration* in order to produce easily comparable percentage values. The following subsections discuss the test suite parameters in detail.

5.2.1 Index configurations

Four different types of index structure have presented in the previous chapters. Section 2.4 has introduced the DataGuide, which is the basis of the new approaches developed in the course of this work. In section 3.2.3, the ID-Comparison CADG and the Signature CADG have been proposed. The latter comes in two variations, the Signature Generation CADG and the Signature Look-Up CADG, as explained in section 4.2.3. The DataGuide, ID-Comparison CADG, and both Signature CADGs have been implemented from scratch to get comparable performance results. Besides, four optimizations applicable to all index structures have been discussed in section 3.3.

To assess the effect of individual optimizations for the different index types, all 64 combinations, or *index configurations*, have been tested with the same sets of queries on the same document collections. Table 5.1 shows the index configurations with their respective IDs, which are used throughout this chapter and also appear in the performance tables in appendix A. There are four rows for the different index types, and sixteen columns representing all combinations of the four optimization techniques. Henceforth the index type and optimizations characterizing an index configuration will be referred to as *tuning paramters*.

	+DB intervals								−DB intervals							
	+indirect keyword occ.				−indirect keyword occ.				+indirect keyword occ.				−indirect keyword occ.			
	+index int		−index int		+index int		−index int		+index int		−index int		+index int		−index int	
	+lab	−lab	+lab	−lab	+lab	−lab	+lab	−lab	+lab	−lab	+lab	−lab	+lab	−lab	+lab	−lab
ID	0	4	8	12	16	20	24	28	32	36	40	44	48	52	58	60
SG	1	5	9	13	17	21	25	29	33	37	41	45	49	53	57	61
SL	2	6	10	14	18	22	26	30	34	38	42	46	50	54	58	62
DG	3	7	11	15	19	23	27	31	35	39	43	47	51	55	59	63

ID: *ID-Comparison CADG*
SG: *Signature Generation CADG*
SL: *Signature Look-Up CADG*
DG: *DataGuide*

DB intervals: *interval encoding on document nodes*
indirect keyword occ.: *indexing of indirect keyword occurrences*
index int: *interval encoding on index nodes*
lab: *secondary label index*

Table 5.1: Index configurations

5.2.2 Document collections

doc. collection	content	size (MB)	# nodes	# paths	# keywords	depth	recurs.
Cities	German cities	1.3	16448	253	18992	7	no
XMark	XML Benchmark	29.2	417160	515	84235	13	yes
NP	German noun phrases	509.2	4585321	2349	129534	40	yes

keywords: *number of different keywords occurring in the documents*
paths: *number of different label paths in the documents*
nodes: *number of document nodes*
recurs.: *recursive paths (i.e. paths with repeated labels)*

Table 5.2: Document collections used for evaluation

Table 5.2 lists the three document collections used for this performance evaluation.[25] The first one, *Cities*, contains manually compiled information about seven German cities. It is relatively small and fairly balanced in several respects: there are no overly long document paths, most leaf nodes contain a limited amount of different keywords (about a dozen on average), and the DTD is not recursive. The second collection, *XMark*, has a far richer content, many leaf nodes containing several hundred different keywords. There are also a lot of modestly recursive paths in the documents. On average, *XMark*'s paths are a few steps longer than those in *Cities*. *XMark* has been generated by the XML Benchmark Project generator [XML]. The third document collection, *NP* [OMM98], is a corpus comprising thousands of syntactically analyzed German noun phrases.[26] As such it is much more structure-oriented. Not surprisingly for a syntax tree, the vast majority

[25] For some statistics about the index structures built over the different document collections, see table 5.8 on page 78 and table 5.9 on page 81.

[26] The aim of a syntax analysis is to construct a hierarchy of syntactical components reflecting the structure of a given phrase, e.g. a sentence or a noun phrase. To this end, the phrase is divided into its atomic constituents (i.e., for the most part, individual words) which are then assigned to the corresponding components (e.g. determinator or noun). This procedure recursively reaches ever higher levels of abstraction.

of its leaf nodes contain only a single keyword. Yet the document paths are often extremely long (up to fourty steps). This is mostly due to the heavily recursive structure of noun phrases.

In table 5.2 above, the number of label paths covers not only root-to-leaf paths, but also the label paths of inner document nodes. Hence it is equal to the number of nodes in the corresponding index tree.

5.2.3 Queries

5.2.3.1 Query classification

To facilitate the comparative evaluation of the various performance tests, a systematic and flexible query classification scheme has been developed in the course of this work. Although designed to fit in with the performance tests for the CADG, it is sufficiently generic to serve other evaluations related to tree queries.

In the tests carried out for this performance analysis, the evaluated queries are classified according to seven criteria, or *query characteristics*, which are most relevant to the tuning parameters (see section 5.2.1 above). As shown in table 5.3 below, each criterion is represented by a Boolean value indicating the presence or absence of a specific characteristic for all queries in a given class. (The query characteristics are described in more detail below.) These Boolean values characterizing the queries in any given class together form a seven-bit string called *query class signature*, which is used to identify individual query classes in a meaningful manner. For instance, the class containing queries where all seven characteristics are present has the query class signature `1111111`. Much more than alternative representations such as 127, query signatures can serve as a mnemonic indicating which kind of query the corresponding query classes contain. Throughout the following discussion, and in some figures, individual query classes are therefore referred to by their signatures.

nr.	values	meaning	threshold
6	`0------`	matching query	
	`1------`	mismatching query	
5	`-0-----`	query with many path joins	$0.3 * (n_{q_s} + n_{q_t})$
	`-1-----`	query without few path joins	
4	`--0----`	query with many soft-edged structural nodes	$0.3 * n_{q_s}$
	`--1----`	query with few soft-edged structural nodes	
3	`---0---`	query with few selective labels	$0.3 * n_{q_s}$
	`---1---`	query with many selective labels	
2	`----0--`	query with many soft-edged textual nodes	$0.3 * n_{q_t}$
	`----1--`	query with few soft-edged textual nodes	
1	`-----0-`	query with few path-selective keywords	see table 5.4
	`-----1-`	query with many path-selective keywords	
0	`------0`	query with few node-selective keywords	see table 5.4
	`------1`	query with many node-selective keywords	
n_{q_s} : *number of structural query nodes* n_{q_t} : *number of textual query nodes*			

Table 5.3: Query characteristics for classification

Table 5.3 lists the seven query characteristics employed for query classification. First, it has been observed that the evaluation performance for a given query is affected by the number of matches retrieved. The reason is that the evaluation of a mismatching query may be aborted as soon as the mismatch becomes obvious, which possibly saves the processing of huge parts of the query tree.[27] Besides, the time it takes to create a data structure for the retrieved hits (e.g. a Complete Answer Aggregate, as described in section 4.1.3) usually varies with the size of the answer. Hence the first query characteristic distinguishes maching and mismatching queries.[28] The second characteristic concerns the number of path joins required in retrieval phase 4. Recall from section 3.3.2 that structural node identification of document nodes strives to enhance path joining. Since this phase is skipped when evaluating path queries, the index configurations 0 through 31 should only profit from interval encoding on document nodes when evaluating non-trivial tree queries. The characteristics with the bit numbers 4 and 3 deal with a query's tendency to entail index navigation during evaluation. The former reflects the number of soft-edged structural nodes in a query tree, whereas the latter depends on the number of unselective labels. Both are important when using a label index (see section 3.3.4). Characteristic number 2 is the analogon of 4 for textual query nodes. It matters mainly for configurations which index indirect keyword occurrences (see section 3.3.1). Finally, query keyword properties are encoded in the two remaining characteristics. While number 1 stands for the path selectivity of the keywords in the query tree, number 0 regards their node selectivity. The queries listed in appendix A are homogeneous with respect to these keyword properties, i.e. each contain only keywords which share the same path and node selectivity characteristics.

The threshold terms given in the last column of table 5.3 quantify the query characteristics just described. For instance, to be classified as join-intensive, a given query needs to have at least 30% path join nodes, i.e. nodes with a fan-out greater than one. For obvious reasons, the concept of a threshold does not apply to characteristic 6. The keyword properties (characteristics 1 and 0) are quantified individually for each document collection, using two thresholds instead of a single one as follows. A keyword is considered node-selective if its total number of occurrences in the document collection does not exceed a lower threshold $t_{sel,n}$. Keywords occuring more than $t_{unsel,n}$ times ($t_{sel,n} < t_{unsel,n}$) are node-unselective. Analogously, two thresholds $t_{sel,p}$ and $t_{unsel,p}$ are fixed to quantify path selectivity. Table 5.4 shows the empirical threshold values used for the three document collections during experimental evaluation[29], which have been tuned manually for each collection. The numbers in parentheses indicate the corresponding fractions of paths and nodes, respectively. The total path and node count for all document collections is given in table 5.2 above.

threshold	*Cities*		*XMark*		*NP*	
$t_{sel,p}$	$0.03 * p$	(8 paths)	$0.005 * p$	(3 paths)	$0.0022 * p$	(5 paths)
$t_{unsel,p}$	$0.055 * p$	(14 paths)	$0.1 * p$	(52 paths)	$0.013 * p$	(31 paths)
$t_{sel,n}$	$0.004 * n$	(66 nodes)	$0.0004 * n$	(167 nodes)	$0.00033 * n$	(1513 nodes)
$t_{unsel,n}$	$0.016 * n$	(263 nodes)	$0.005 * n$	(2086 nodes)	$0.003 * n$	(13756 nodes)

p: *number of label paths in the document collection*
$t_{sel,p}$: *lower path selectivity threshold*
$t_{unsel,p}$: *upper path selectivity threshold*
n: *number of nodes in the document collection*
$t_{sel,n}$: *lower node selectivity threshold*
$t_{unsel,n}$: *upper node selectivity threshold*

Table 5.4: Keyword selectivity thresholds

[27] Currently structural mismatches (i.e. queries which cannot be satisfied due to non-existant labels, missing paths, etc.) and textual mismatches (i.e. queries specifying keywords which do not occur in the document collection) are not distinguished.

[28] The corresponding bit in the query class signature is the left-most one. The bit numbers shown in table 5.3 beginning with the least significant bit (i.e. the right-most one, according to the binary representation of integers), the left-most bit has the number 6.

[29] Note, however, that for the two manually compiled query sets (see the next subsection), much stricter criteria have been applied: the query keywords selected are always well beyond the given thresholds to contrast the different query classes.

5.2.3.2 Query generation

Each of the four test suites accomplished in the course of this work has its own set of queries to be evaluated. There are two hand-crafted and two automatically generated sets. Both generation methods have their pros and cons: while manually devised queries allow for systematic comparison and exact classification, they are tedious to produce and are usually created with a certain idea or expectation in mind, which may affect the choice of index features they actually test. By contrast, random query generators are normally more objective, making it easy to create extensive test query sets. However, automatic query classification is error-prone and hard to monitor for large sets of queries. Besides, for some query classes no generated queries could be used in this evaluation, for several reasons. Sometimes the generated queries were too general, such that evaluating them repeatedly was infeasible given the number and the size of the test suites to accomplish. Occasionally classes were skipped because the generator failed to create suitable queries in a fixed number of attempts. There are certain combinations of structure and content characteristics which are hard to fulfill, e.g. when keywords with the required characteristics only occur in document nodes below a certain level. A typical case are the query classes with one of the last two bits set, but not both: they require keywords which are node- but not path-selective, or vice versa. Keywords with these properties are hard to find in most document collections.

Depending on the query generator's implementation, further problems may arise. For example, the generator used for this performance evaluation creates queries based on a given DTD, rather than on the index tree for a document collection. As a consequence, when a document collection is sparse with respect to the space of valid documents spanned by its DTD, mostly structural mismatches are generated. For obvious reasons, this is the case for the document collection *NP* (see section 5.2.2): first, the noun phrases it contains are taken from natural language sources, where many valid phrases almost never occur because they are too complex. Second, the DTD underlying *NP* is recursive, and as such spans an infinite space of valid documents.

To capture the advantages of both manual and automatic query generation while compensating for their respective drawbacks, all index configurations have been tested with two query sets of either kind. Table 5.5 lists the four test suites and some statistical information about them. *CitiesMan*, being the most complete and systematic one, serves as a basis for the evaluation in section 5.4. Its hand-written queries are well-classified and cover all query classes. *CitiesAuto* provides twice as much queries, backing *CitiesMan* with random control data. It covers less query classes than the former, and contains misclassified queries. *XMarkAuto* and *NPMan* deal with the two remaining document collections. Since they are considerably larger than *Cities*, these two test suites are useful for assessing scaling effects. However, both leave a considerable amount of query classes blank. Especially *NPMan* is very incomplete and features only highly selective keywords (due to the path reconstruction problem, see section 5.4.4). As a consequence, its query set is overly tailored to CADGs. But even for the more complete test suites, one needs to keep in mind that query classes are not necessarily equally distributed in real-world retrieval scenarios. Hence the plots of average results over all queries presented in the remainder of this chapter only serve vizualization purposes and should be interpreted cautiously. The actual analysis is based on the raw data given in appendix A.

test suite	document collection	query generation	# classes	# queries	mismatches (%)
CitiesMan	*Cities*	manual	128	166	50
CitiesAuto	*Cities*	automatic	126	333	55
XMarkAuto	*XMark*	automatic	76	163	67
NPMan	*NP*	manual	36	61	62
128 query classes total					

Table 5.5: Test suites

Table 5.6 gives the relative portions of queries featuring a given query characteristic for all tests suites. Recall from table 5.3 above that a value of 0 for any query characteristic represents a less selective or more difficult setting than a value of 1. For instance, the query set of the *CitiesMan* test suite contains 50% matching queries (characteristic 6) and 46% tree queries (characteristic 5).

test suite	queries with a given characteristic (%)						
	0------	-0-----	--0----	---0---	----0--	-----0-	------0
CitiesMan	50	46	59	59	46	53	53
CitiesAuto	45	45	48	50	49	51	50
XMarkAuto	33	37	45	47	37	46	34
NPMan	38	11	58	39	16	31	13

Table 5.6: Query set statistics

5.3 Infrastructure

All four test suites have been carried out sequentially on the same computer (AMD Athlon XP 1.33 GHz, 512 MB, running SuSE Linux 7.3 with kernel 2.4.16), each one in a single run. The testbed system described in section 4.1 has been used with a Java HotSpot Client Virtual Machine (Java 2 Runtime Environment, Standard Edition, build 1.4.1_01-b01) and the PostgreSQL relational database system, version 7.1.3. Client application and database server were running on the same computer to minimize the influence of data transmission on performance measuring. The database cache has been disabled. To lessen file system cache effects, each query has been processed once without taking the results into account. The following three iterations of the same query have been averaged to produce the results given in appendix A. Since this way the time to fetch blocks from disk is minimized, the CADG's performance gain over the ordinary DataGuide due occurrence fetching might actually be even larger than the test results suggest. However, this issue has not been investigated.

5.4 Retrieval performance

This section deals with the performance results obtained for the experimental setting and the evaluation environment described above. First, a brief overview over all test suites is given, based on the average performance over all query classes. Second, the results of the four index types are examined and compared to each other in more detail. Afterwards the strengths and weaknesses of the four optimization techniques are assessed. The results are summarized in the last part of this section, which also provides recommendations and tentative guidelines for choosing an index configuration for a given retrieval scenario.

The observations and conclusions presented in this section are mainly based on the raw performance data given in appendix A. For convenience, different plots of selected portions of these data are provided. Furthermore, primitive data mining techniques have been applied to extract statistical data about the preferred query classes of any index configuration, as well as dependencies between query classes and individual tuning parameters. As a means to amplify and visualize trends and patterns in the extensive data obtained in the four test suites, these techniques are necessarily less exact, relying largely on aggregated (averaged) and ranked data. To verify the following observations, the reader is always implicitly referred to the complete data set in appendix A.

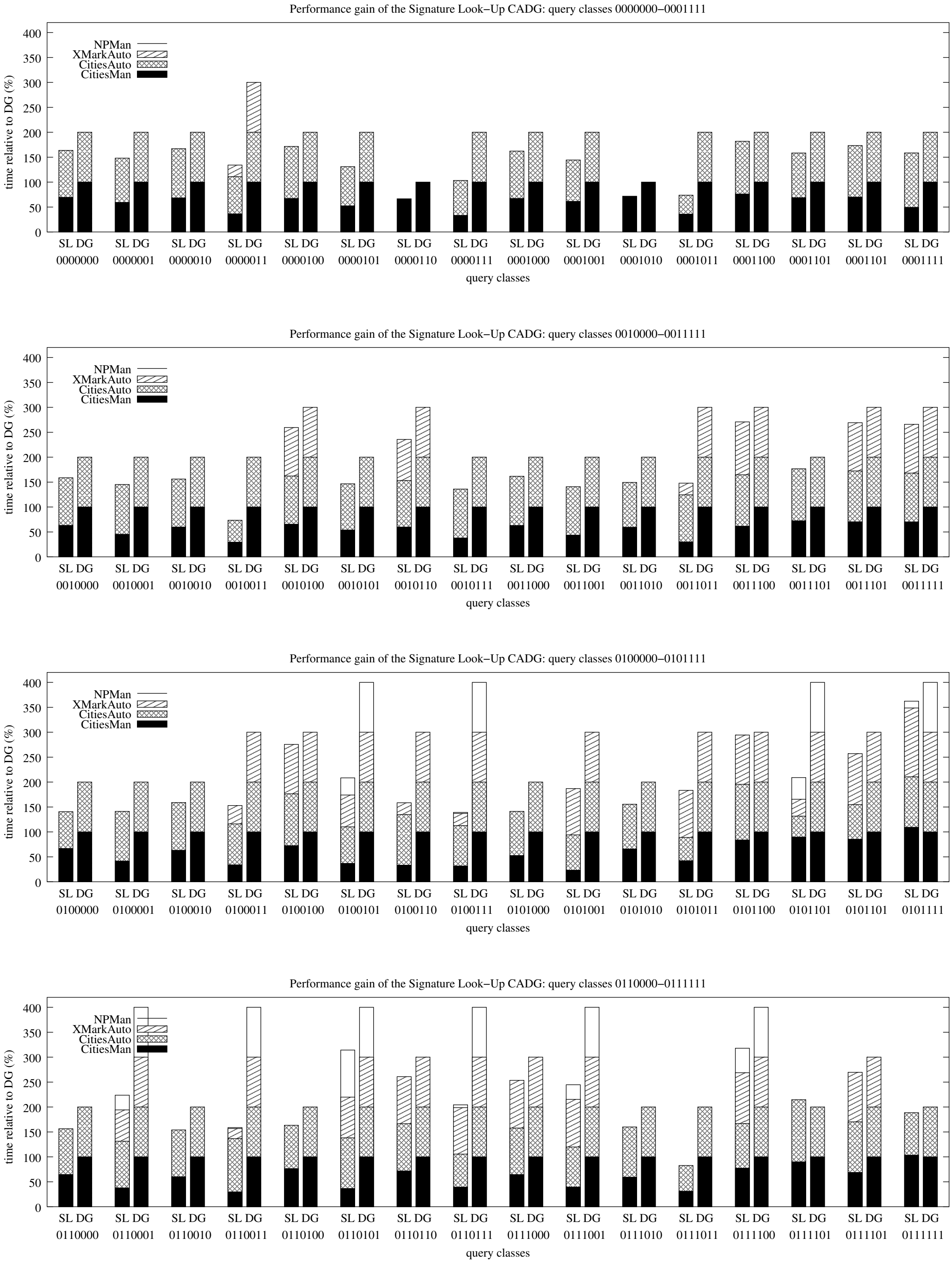


Figure 5.2: Performance gain Signature Look-Up CADG vs. DataGuide (match queries)

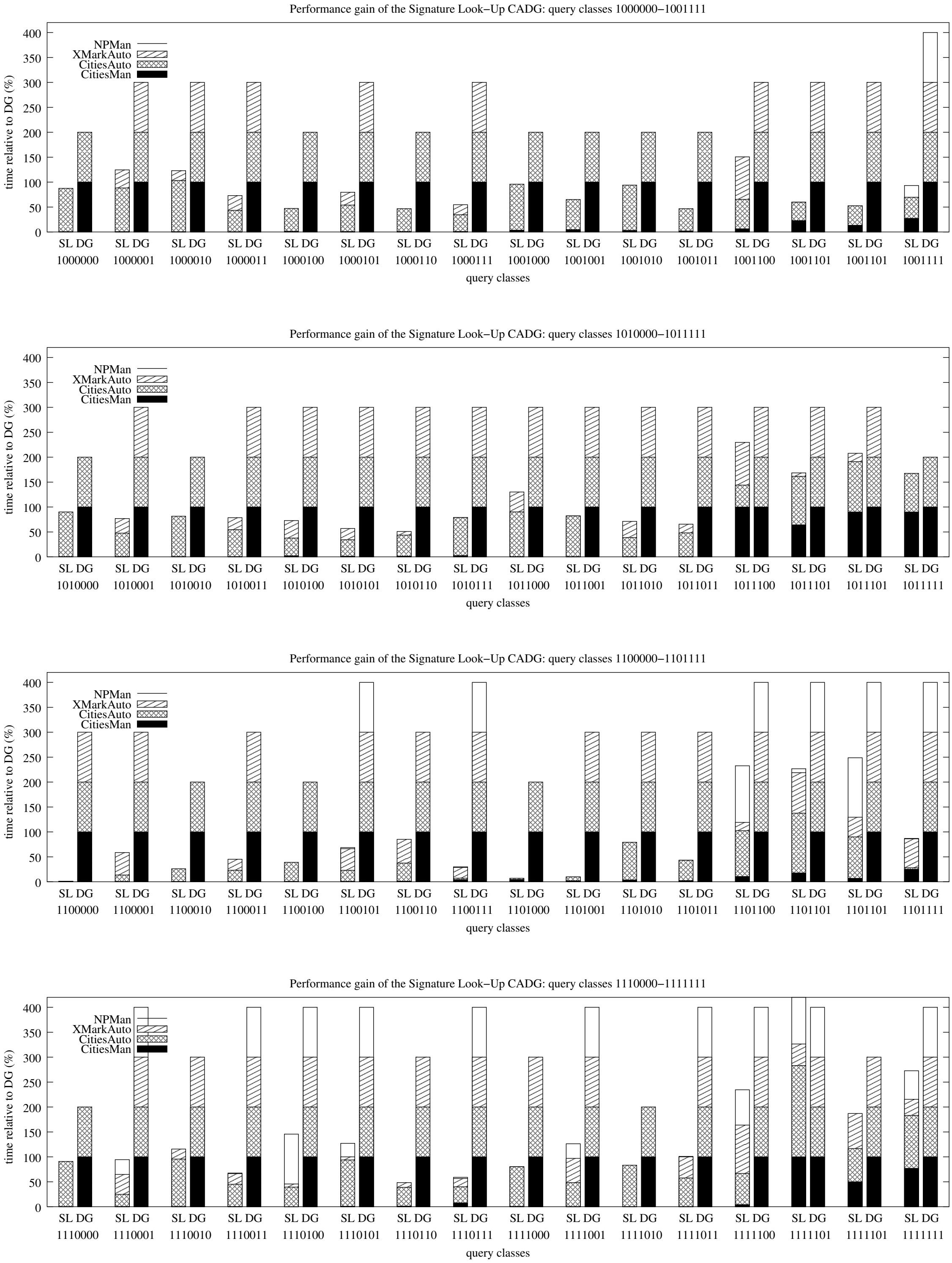


Figure 5.3: Performance gain Signature Look-Up CADG vs. DataGuide (mismatch queries)

5.4.1 Overview of the test suites

Figures 5.2 and 5.3 contrast the average time needed to evaluate a query from a given class with either the Signature Look-Up CADG or the ordinary DataGuide in all four test suites. (The other two CADGs appear in subsequent charts.) Each pair of boxes in the charts represents the two index structures applied to all queries in the corresponding class. No optimization is used with either index, i.e. the data are obtained from the index configurations 62 and 63, respectively. The different box segments show the results of the four test suites. Note that a box may consist of less than four segments whenever the corresponding query class is not present in one or more test suites. However, there is always the same number of segments in either box in a pair, i.e. the corresponding results are presented for both index structures. According to the description above, *CitiesMan* covers all 128 query classes, hence there is always a segment representing the results obtained for this test suite (the lowest segment of each box). Seemingly missing segments, like the CADG's *CitiesMan* segment in the left-most pair of figure 5.3's first row, are too small to be displayed in the given scale. For comparison, the DataGuide's segments have all been assigned a value of 100% (right box of a pair). The Signature Look-Up CADG's corresponding segment values have been computed relative to the DataGuide's segments (left box of a pair).

There are several regularities to be observed in the two figures. First, for almost all query classes, the Signature Look-Up CADG's overall performance over all test suites is better than the DataGuide's. In figure 5.3, the performance gain is much greater than in figure 5.2 (for *CitiesMan* by several orders of magnitude). In other words, the Signature Look-Up CADG is especially fast for most mismatch queries. However, there are salient differences between the four test suites. While *CitiesMan* almost always shows a huge performance gain over the DataGuide (up to 70% for match queries, and much more for mismatches; all in all mostly more than 50%), *CitiesAuto* is much harder for the Signature Look-Up CADG. There are individual match query classes with a considerable performance gain, and few where the DataGuide is faster than its competitor. However, the performance results of two indices are much closer in this test suite, as far as match queries are concerned. For mismatching queries from *CitiesAuto*, the Signature Look-Up CADG is usually much better than the DataGuide.[30] Apparent exceptions are partly due to the small absolute evaluation needed for most mismatching queries. For instance, the difference between the Signature Look-Up CADG and the DataGuide in the antepenultimate pair in figure 5.3's last row amounts to only 1.67 milliseconds. The same phenomenon can be observed for *NPMan*. As a first result, one may state that the automatically generated query sets are much more of a challenge to the CADG. Furthermore, the CADG's performance gain is smallest for queries with little navigation: in figure 5.2, the last four pairs (representing queries with few soft edges and selective labels) in all rows show less difference between the two *CitiesMan* results. In figure 5.3, the effect is even more salient, although restricted to the first and third row.

In the following, the performance results obtained for each of the four test suites are plotted in a separate box chart. Each box depicts the average time (in milliseconds) needed to evaluate a query from any given query class. The sixty-four index configurations (see section 5.2.1) are displayed in groups of four, each group representing four index configurations with the same optimizations but a different index type. The first row in each figure covers all index configurations with interval encoding on document nodes, whereas the second row shows results for the configurations without this optimization. For orientation, keep in mind that when "reading" a figure from left to right and top to bottom, the index configurations become less optimized, as indicated by the description below each group of boxes. Also remember that for each index configuration, the average result over all queries in the test suite is shown. This does not imply that all query classes covered by any of the test suites are equally important in real-world retrieval scenarios. Plots for individual query classes are added in the following sections, where necessary. The complete test results can also be found in appendix A.

[30] As discussed in section 2.4.3, the DataGuide's retrieval algorithm may privilege either structural or textual mismatches, but not both. In the implementation underlying this performance evaluation, the first option has been chosen.

5.4.1.1 *CitiesMan*

The performance results obtained from the first test suite, *CitiesMan*, are shown in figure 5.4. As can be seen from the figure above, the average performance gain of the three CADGs over the ordinary DataGuide is close to 50% for all combinations of optimizations. Ranking the index types for each such combination (i.e. each group of four boxes) yields a characteristic pattern which is present throughout the figure: the Signature Look-Up CADG is always best, followed by the ID-Comparison CADG and the Signature Generation CADG in relatively small distances. The DataGuide is constantly outperformed by the three CADGs. The retrieval performance of all four index types is enhance considerably when indirect keyword occurrences are indexed. As can be seen from the difference between the left and right halves in either row, this optimization speeds up retrieval by another 50% on average.

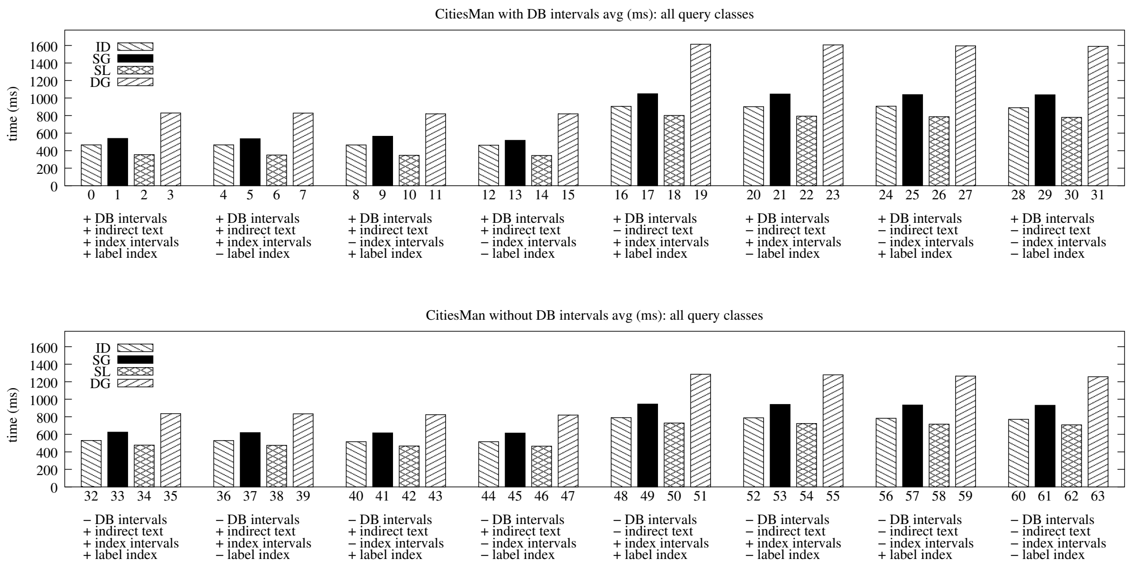


Figure 5.4: Performance results for test suite *CitiesMan* (average over all query classes)

An interesting point to observe is the effect interval encoding of document nodes has on the DataGuide: surprisingly retrieval with plain document node IDs is faster than with interval-encoded ones, unless indirect keyword occurrences are indexed. To anticipate the discussion of this phenomenon in section 5.4.3.2, processing query paths with soft-edged textual nodes when only direct keyword occurrences have been indexed may entail considerable navigation in the index tree, which in turn causes occurrences to be fetched for many index nodes. Unlike the CADG, occurrence fetching for the DataGuide involves two instead of one database access. However, the storage overhead introduced by interval encoding on document nodes slows down the fetching process. The benefit of interval-encoded document nodes, which is supposed to compensate for this drawback, comes into effect only when querying non-trivial tree queries. Yet such queries are less frequent in *CitiesMan* than e.g. in *CitiesAuto*, where the opposite phenomenon can be observed (see section 5.4.1.2).

5.4.1.2 *CitiesAuto*

Figure 5.5 below illustrates the results for the *CitiesAuto* test suite. Compared to *CitiesMan*, the difference between the CADGs and the DataGuide is reduced, especially when document nodes are not interval-encoded (second row). This is probably due to the fact that *CitiesAuto* features far more large matching tree queries than the other test suites. In *CitiesAuto*, 41% of the tree queries match, in contrast to 22% in *XMarkAuto*. In the former suite, 80% of the matching tree queries contain more than six nodes, compared to only 38% in the latter and 32% in *CitiesMan*. This explains why the apparent negative effect of document node intervals on the DataGuide (see section 5.4.3.2), which can be observed in both *CitiesMan* and *XMarkAuto*, is reversed in *CitiesAuto*: here many more queries are tested where interval encoding on document nodes actually speeds up retrieval.

By contrast, all four index structures are equally affected when the optimization is missing, which brings their performance results closer together, as can be seen in the second row of figure 5.5. The absolute difference between the Signature Look-Up CADG still amounts to 150-200 milliseconds in average, but is considerably smaller than in the case of *CitiesMan*. The characteristic pattern from the first row, which can be observed in all four test suites, is overlapped by a general performance loss induced by the numerous database operations needed for path joining and, in case structural soft edges are involved, path reconstruction (see section 5.4.4). Due to this base latency, the CADG's performance gain appears less important than it would given a more efficient path join technique. Note that path joining and reconstruction techniques are more part of the evaluation algorithm than the index structure. The *CitiesAuto* suite therefore shows more than any other test suite how much the overall performance of the retrieval system depends on the query processor using the index. In other words, this test suite actually says more about the query evaluation algorithm than about the index structures being tested. Section 5.4.4 briefly discusses possible approaches to the path reconstruction problem.

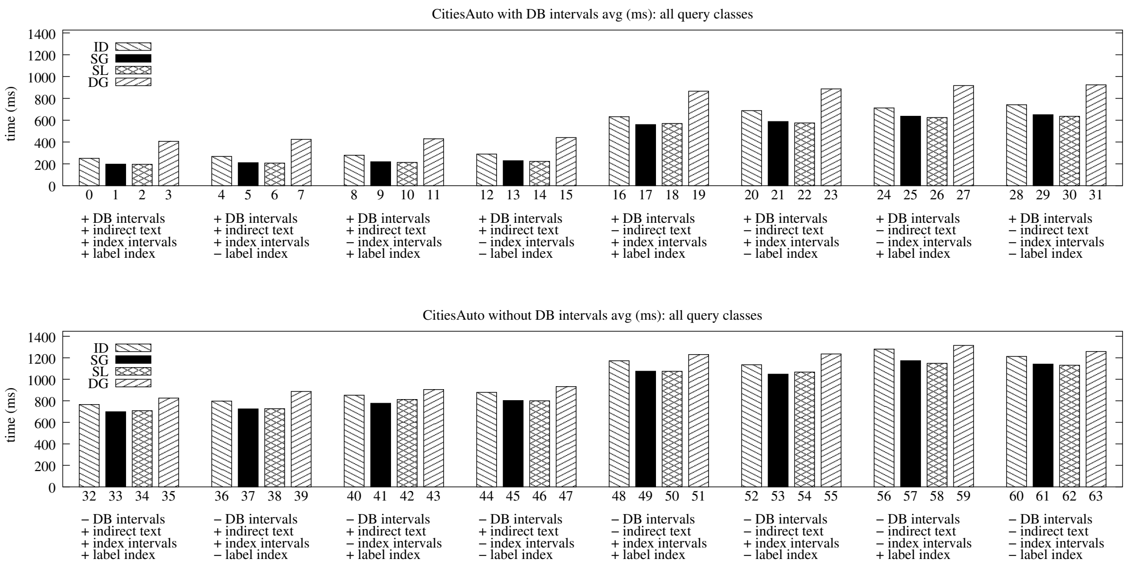


Figure 5.5: Performance results for test suite *CitiesAuto* (average over all query classes)

Note that the scales in figure 5.5 and figure 5.4 on page 71 are slightly different. The ranking among the three Content-Aware DataGuides has also changed, the Signature Generation CADG being constantly faster than the ID-Comparison CADG, and sometimes even outperforming the Signature Look-Up CADG. Besides the remarkable performance gain caused by structural node identification on document nodes, indirect keyword occurrence indexing proves very effective, like for *CitiesMan*.

5.4.1.3 *XMarkAuto*

The third test suite, *XMarkAuto*, produced the results shown in figure 5.6. Again a considerable difference in retrieval performance is entailed by interval encoding on document nodes, like for the *CitiesAuto* test suite. Once more this effect is restricted to DataGuide configurations. The other index types are mostly invariant to this and any other optimization technique. Only the indexing of indirect keyword occurrences, which has been shown to have a major influence on the first two test suites, entails a noticeable difference (note that the scale of figure 5.6 is more than three times as large as for the previous plots, which makes the difference seem less important than it actually is). These observations may be explained mainly by scaling effects, as discussed in section 5.4.3.2.

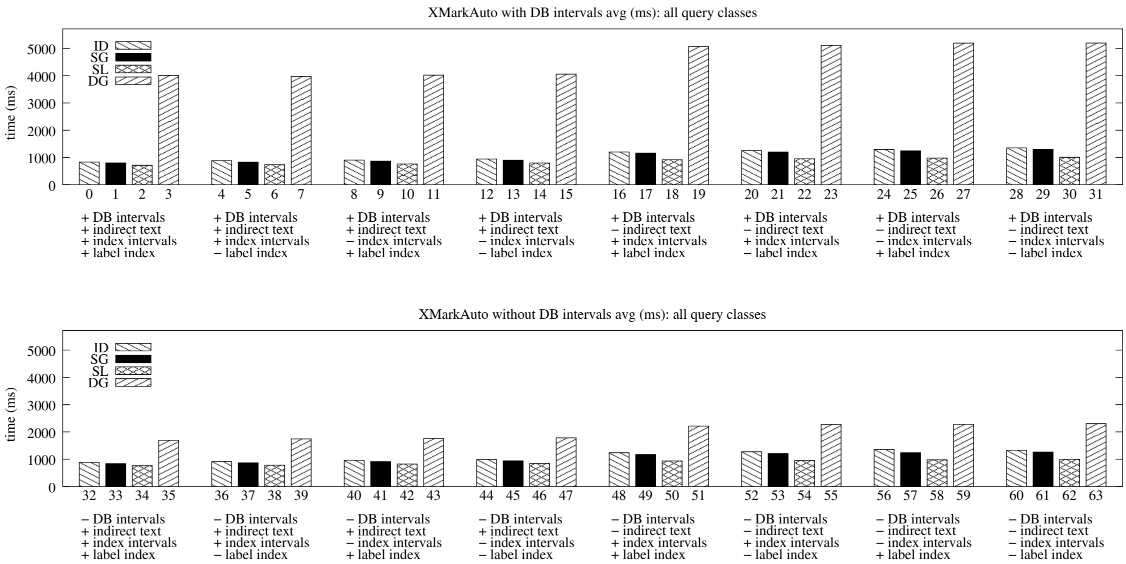


Figure 5.6: Performance results for test suite *XMarkAuto* (average over all query classes)

5.4.1.4 *NPMan*

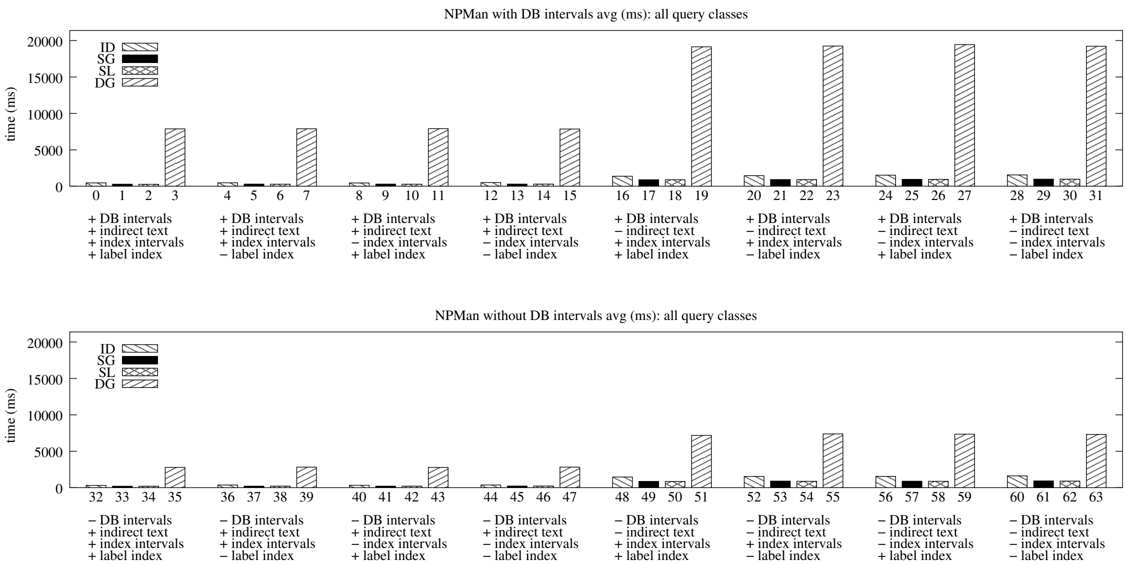


Figure 5.7: Performance results for test suite *NPMan* (average over all query classes)

The most challenging, yet most incomplete test suite is *NPMan*, whose performance results are plotted in figure 5.7. It is distinguished by an extremely poor performance of the DataGuide, compared to all three CADGs. This is partly due to the limited choice of query classes generated for this huge document collection, as mentioned in section 5.2.3.2. Since queries with unselective keywords were often hard to evaluate on a reasonable time scale, mainly because of the path reconstruction problem described in section 5.4.4, there are far more queries with selective keywords, which clearly privileges the Content-Aware DataGuide. Comparing the different CADGs to each other reveals a similar pattern as for the *CitiesAuto* and *XMarkAuto* test suites, although attenuated by the much larger scale (increased by factor 10 in figure 5.7).

5.4.2 Index types

The following subsections evaluate the above performance results in more detail, each focussing on a single tuning parameter. First the four index types are examined and compared to each other. The rest of this section is dedicated to the four optimization techniques described in section 3.3.

5.4.2.1 Signature Look-Up CADG

As can be seen from the plots in the previous section, the Signature Look-Up Content-Aware DataGuide performs best for a wide variety of query classes and tuning parameter settings. A ranking of index configurations according to their evaluation performance (see table 5.7) reveals that in *CitiesMan*, combinations involving the Signature Look-Up CADG appear most often among the best three configurations for a given query class (column total). As discussed below, this is largely due to its excellent performance for mismatching queries. Moreover, the Signature Look-Up CADG frequently prevails over the other index types, holding all three top ranking positions in configurations with different optimizations (column exclusive). This trend is confirmed by *XMarkAuto*'s results: there the Signature Generation CADG also joins the top group of configurations for matching queries very frequently, but is outperformed by the Signature Look-Up CADG for mismatches. In the case of *CitiesAuto*, the Signature Look-Up CADG falls back behind the second signature-based index, but still beats the ID-Comparison CADG by far. In *NPMan*, the latter clearly predominates.

test suite	query	appearances among the best three index configurations							
		ID		SG		SL		DG	
		total	excl. (%)	total	excl. (%)	total	excl. (%)	total	excl. (%)
CitiesMan	match	44	89	6	0	19	74	1	0
	mismatch	23	13	8	100	53	60	0	0
CitiesAuto	match	23	0	65	5	50	0	6	17
	mismatch	11	9	44	25	46	26	1	0
XMarkAuto	match	19	0	29	0	29	0	6	0
	mismatch	4	0	27	11	42	45	2	0
NPMan	match	11	64	6	17	2	0	1	0
	mismatch	14	21	10	10	4	50	5	0

ID: *ID-Comparison CADG* SG: *Signature Generation CADG*
DG: *DataGuide* SL: *Signature Look-Up CADG*
excl.: *number of query classes with no other index type among the best three configurations*

Table 5.7: Performance ranking of index structures (sum over all query classes)

The Signature Look-Up CADG's strongest competitor in this ranking is the ID-Comparison CADG, which is even better for match queries. Clearly the Signature Look-Up CADG achieves its greatest performance gain over the other index structures when evaluating mismatching queries.[31]

[31] This is a useful feature in real-world applications, although it could not compensate for poor retrieval of matches. But the synopsis of all four test suites in figure 5.2 shows that the Signature Look-Up CADG considerably speeds up query processing for a wide variety of match classes.

In case of a textual mismatch, the Signature Look-Up CADG is considerably faster than any other content-aware approach, let alone the DataGuide, for obvious reasons: any keyword not occurring in the document collection is identified by the Signature CADG during the look-up in its signature table. Thus the mismatch is recognized immediately in retrieval phase 0, without entering the navigation phase. By contrast, the Signature Generation CADG creates signatures for any keyword, discovering only during retrieval phase 1 that a mismatching keyword is being queried. Finally, although the ID-Comparison CADG's behaviour is similar to the Signature Look-Up CADG's, it is mostly even slower than the Signature Generation CADG when processing textually mismatching queries. As described in section 4.2.3, the look-up of relevant index nodes is carried out on the content/annotation table, which is indexed internally using a 2-dimensional key instead of a 1-dimensional as in the case of the signature table. As a result, the ID-Comparison CADG's look-up takes considerably longer than the Signature Look-Up CADG's. As mentioned earlier, maintaining a separate index node look-up table for the ID-Comparison CADG (analogously to the Signature Look-Up CADG's signature table) might attenuate this effect.

The benefit of signature-based content-aware navigation becomes apparent when examining which query classes are evaluated fastest for different index configurations. In the *CitiesMan* test suite, query classes with soft structural edges appear among the three best-ranked classes only for Signature Look-Up CADG configurations, except when a label index is used. The same is true for *CitiesAuto* and *XMarkAuto* although there indexing of indirect keyword occurrences also allows some configurations to have such query classes among their best-supported ones. Not surprisingly, the performance gain induced by content-aware navigation is greatest when querying highly selective keywords, as illustrated for the sample query classes `0100111` (selective query keywords) and `0100100` (unselective query keywords) in figure 5.8. Here the Signature Look-Up CADG gains nearly a factor 4 compared to the DataGuide, which does not take advantage from high keyword selectivity.

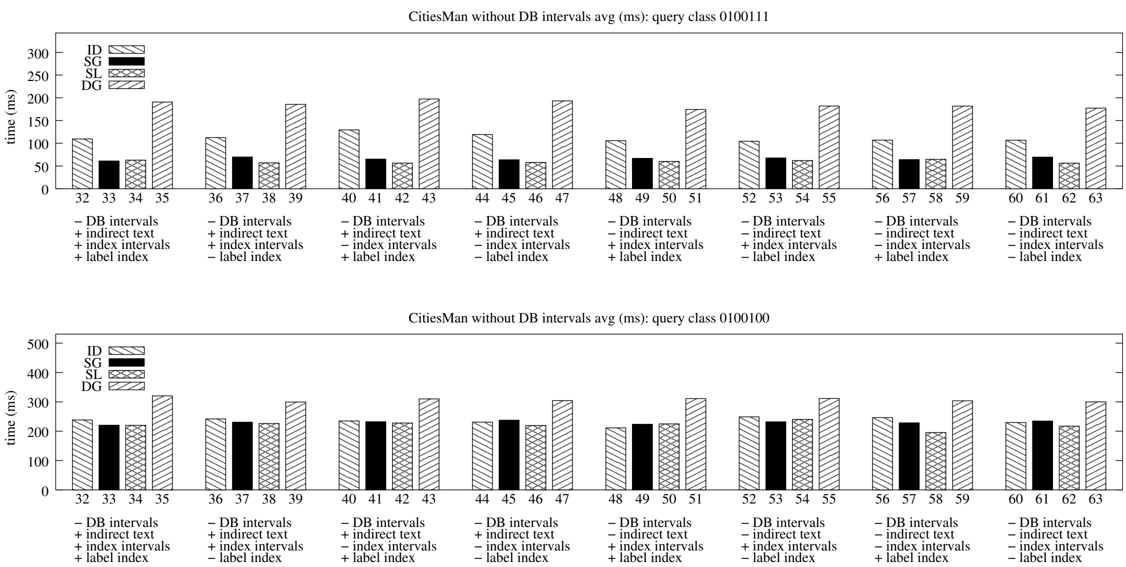


Figure 5.8: Performance gain of content-aware navigation for selective and unselective keywords

But also queries with soft-edged textual nodes can profit from content-aware navigation, provided that indirect keyword occurrences are not indexed. In *CitiesMan*, the Signature Look-Up CADG is the only index having such queries among its best-supported three query classes under these circumstances. In *CitiesAuto*, even indexing of indirect keywords occurrences is not enough to make other index configurations support textual soft edges equally well: here only Signature Look-Up CADG configurations have such classes among their privileged ones, regardless of whether indirect keyword occurrences are indexed or not. Also in *XMarkAuto*, the Signature Look-Up CADG shows a strong inclination towards these query classes. *NPMan*, with its much larger index tree, differs in this respect; here the occurrence fetching optimization is the only decisive factor when evaluating queries with soft-edged textual nodes.

5.4.2.2 Signature Generation CADG

As can be seen from the average performance diagrams in figures 5.4 to 5.7, the Signature Generation CADG's evaluation results are very close to those of the Signature Look-Up CADG, although the latter almost always performs a little bit better. In *CitiesMan*, the Signature Generation CADG produces the worst results compared to the other two Content-Aware DataGuides, falling behind the ID-Comparison CADG. Since the two signature-based index structures only differ in their way of obtaining keyword signatures during query preprocessing, the only plausible, though surprising conclusion is that creating signatures from scratch during retrieval is more expensive than fetching ready-made signatures from the database. Of course this also depends on the signature creation algorithm employed by the Signature Generation CADG. The one described in section 4.2.3 not being overly complex, the clear advantage of signature look-up over signature creation is remarkable.

5.4.2.3 ID-Comparison CADG

An interesting aspect of content-aware navigation and occurrence fetching is the question whether the exact, index-node based ID-Comparison CADG is superior or inferior to the two heuristic signature approaches. Based on the performance results presented so far, the answer is quite unambiguous: in all four test suites, the signature-based approach (at least when realized as a Signature Look-Up CADG) outperforms the node-ID based one. Interestingly enough, even when querying document collections which contain many different keywords, such as *XMark*, the Signature CADG's heuristic approach does not degenerate. One could have expected that when many index nodes' signatures are saturated, due to the large sets of keyword occurrences they reference, the Signature CADG may tend to lose its content-awareness. But although the signature-based navigational structures built for the *XMark* collection indeed show saturation effects on the highest levels (with 64-bit signatures and 3 bits set in each keyword signature), the Signature Look-Up CADG (and mostly also the Signature Generation CADG) is clearly superior to the ID-Comparison CADG on average. Note, however, that when considering only the matching queries of the two manually compiled test suites, the ID-Comparison CADG appears more frequently among the best-performing index configurations than the Signature Look-Up CADG (see table 5.7).

When inspecting individual query classes from the *CitiesMan* suite, a characteristic alternation pattern involving the ID-Comparison CADG and Signature Look-Up CADG arises. Extensive parts of the raw data table (see table A.1 on page 92 and table A.2 on page 99) show that while the former performs better than all other index types with the same tuning parameters when given node-unselective query keywords, the latter wins for node-selective keywords. This pattern is almost ubiquitous when indirect keyword occurrences are not indexed, and also persists for configurations with this optimization when there are hardly any textual soft edges in the queries (such that indexing of indirect keyword occurrences is mostly useless). Apparently the phenomenon is related to retrieval phase 1, i.e. navigation. Further examination reveals precise combinations of tuning parameters which allow the Signature Look-Up CADG to outperform its node-ID based counterpart, given node-selective query keywords. For instance, low label selectivity seems to cause more trouble to the ID-Comparison CADG, such that it is beaten by the Signature Look-Up CADG when node-selective keywords are being queried. In other cases, a label index reducing the navigational effort helps the ID-Comparison CADG to regain the leadership for all kinds of query keyword.

To summarize, the less navigation is required during query evaluation, the easier it is for the ID-Comparison CADG to prevail against the Signature Look-Up CADG. This results most probably from the overhead caused by the ID-Comparison CADG's relevance check. Recall from section 3.2.3.1 that for each index node reached during retrieval phase 1, the set of relevant index leaves is examined to find out whether the current path in the index tree can be pruned. For large navigational structures, the ancestor/descendant checks and set operations involved may amount to a considerable overhead compared to the constant time needed for signature-based relevance checks. Apparently this is also true for the relatively small *Cities* collection. However, it remains unclear why the phenomenon is unmistakably related to node selectivity rather than path selectivity (see section 5.4.4 for a brief discussion of this topic). Apart from this seeming irregularity, the influence of the other navigation-related tuning parameters (indexing of indirect keyword occurrences, label index) is as expected. Analogously, the role of the most relevant query characteristics (4: number of soft edges to structural query nodes, 2: number of soft edges to textual query nodes) is not surprising.

5.4.2.4 DataGuide

Throughout the whole experimental evaluation, the DataGuide almost constantly proves to be slower than the three content-aware approaches. As can be seen from figure 5.2 on page 68, there are very few query classes where the DataGuide beats the Signature Look-Up CADG. For mismatch query classes, the result is even more unambiguous (see figure 5.3). The few cases where the DataGuide's performance is comparable to the CADGs' include e.g. queries where a mismatch appears early during retrieval phase 1 or else in phase 2 after hardly any navigation, i.e. when there is little occasion for the DataGuide to get caught in needless path matching or occurrence fetching. But there are also some matching queries, mostly with unselective keywords or none at all, which the DataGuide processes as fast as the CADGs. Figure 5.9 below depicts the performance results for a query class with both path- and node-unselective keywords from the *CitiesAuto* test suite. As a matter of fact, the class `0001100` happens to consist of three queries all of which lack keywords altogether. Hence the three Content-Aware DataGuides face a needless overhead for query preprocessing (which is limited though, since the absence of query keywords keeps the preprocessing phase short), compared to the DataGuide whose brute-force retrieval strategy does not entail any disadvantages here. Accordingly, there is practically no more difference in the performance of the four index structures.

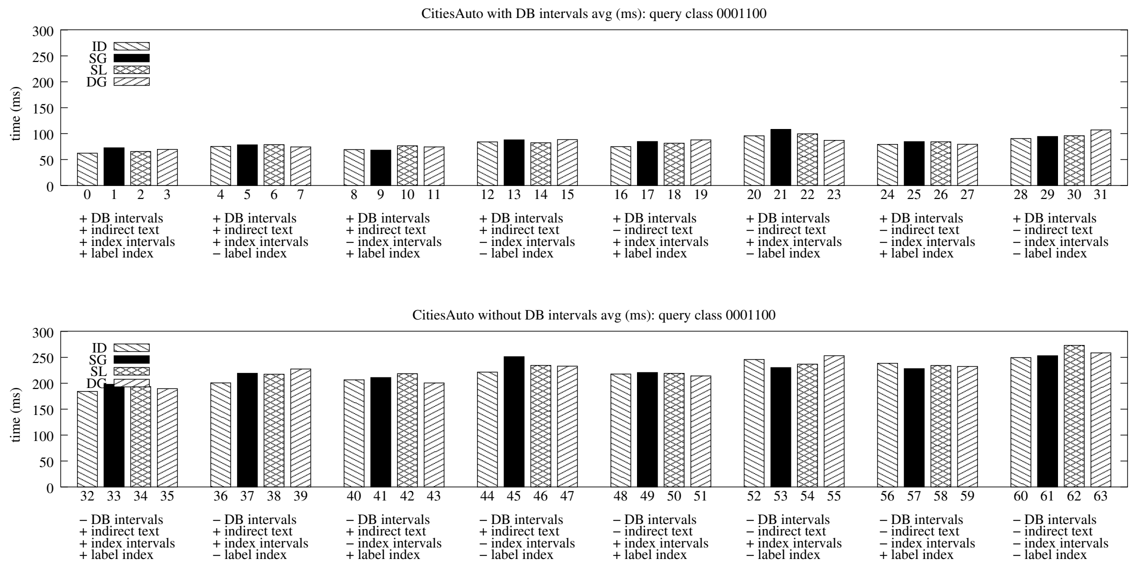


Figure 5.9: Limited benefit of content-awareness for queries without keywords

Among all four index types, the DataGuide's performance depends most on individual query characteristic. In other words, when comparing the results of the same index configuration for query classes which differ in a single characteristic, DataGuide configurations often produce greater differences than those featuring other index types, with no remarkable trend among the query characteristics. Without further analysis, the overall results suggest that the DataGuide's performance easily degenerates for queries which are unselective or more complex in any respect, and is in this sense more closely linked to the characteristics of the queries being processed.

5.4.3 Optimizations

5.4.3.1 Indexing of indirect keyword occurrences

As the figures 5.4 to 5.7 show, indexing indirect keyword occurrences (see section 3.3.1) has a major influence on the retrieval performance of all index configurations. But also the time needed to create index structures with or without support for indirect keyword occurrences, and the space they occupy on disk vary considerably. Table 5.8 lists some approximate statistics for the four test suites used throughout this experimental evaluation. The first three columns show how the retrieval performance is affected when indirect keyword occurrences have been indexed. While the average performance gain in the *CitiesMan* test suite is considerable, other critical factors seem to prevail when processing more complex queries (such as in *CitiesAuto*, see also section 5.4.1.2 for a discussion of this issue) or larger document collections (such as in *XMarkAuto* and *NPMan*). Notwithstanding, for individual query classes the optimization can save a great part of the evaluation time needed to search a subtree of the navigational structure and to fetch occurrences for many index nodes. Due to its duplicate disk access in retrieval phase 2, the DataGuide might be assumed to profit most from indirect occurrence support, which also helps to avoid an exhaustive search in the index tree in case of soft-edged textual query nodes. While this is true when comparing the absolute amount of time saved, the CADGs' relative improvement is on average as high as the DataGuide's.

test suite	performance diff. with ind. occ. (%)			creation time (min.)		storage overhead (%)
	avg. gain	max. gain	avg. loss	+ind. occ.	−ind. occ.	
CitiesMan	75	99	< 10	1.0	0.3	100-400
CitiesAuto	30	90	< 10			
XMarkAuto	20	90	< 10	240-280	45-70	100-300
NPMan	10	99	< 20	190-370	55-75	100-350

+ind. occ.: *with support for indirect keyword occurrences* −ind. occ.: *without support for indirect keyword occurrences*

Table 5.8: Performance with and without support for indirect keyword occurrences

Further statistical analysis reveals that configurations supporting indirect keyword occurrences are more stable, in the sense that their retrieval performance depends less on individual query characteristic than without the optimization. When assembling the three index configurations whose performance in the *CitiesMan* test suite varies most with a given query characteristic, one observes that for most query characteristics, all three positions are held by configurations which lack support for indirect keyword occurrences. The only exceptions are query characteristic more relevant for navigation, namely number 3 (label selectivity) or 4 (number of soft-edged structural query nodes). Analogous regularities hold for the *CitiesAuto* and *XMarkAuto* suites.

5.4.3.2 Structural node identification of document nodes

As mentioned above, the benefits of interval encoding on document nodes can only be observed for the *CitiesAuto* test suite. It features significantly more large matching tree queries than the other three suites. Recall from section 3.3.2 that structural node identification on document nodes is supposed to speed up the path joining phase, whose influence on the overall query evaluation time is reduced with the number of join nodes and vanishes entirely when only a single path is queried. Hence *CitiesAuto* offers much more occasions for this optimization to come into effect, which explains the obvious performance gain to be observed in figure 5.5. On the other hand, interval encoding on document nodes not only introduces a small overhead for managing the more complex node ID objects in the Java application, but also increases the storage required by the content and annotation tables, which slows down occurrence fetching. Especially the DataGuide, whose retrieval phase 2 involves two instead of one database access, is affected by this drawback, which is the price to pay for interval encoding. Figure 5.10 illustrates this phenomenon for a tree query from the *CitiesAuto* test suite. The configurations in the first row are equipped with interval encoding for document nodes, unlike their counterparts in the second row. Obviously all index configurations, including the DataGuide, profit from the optimization; however the three CADGs are less affected by the increased storage. A similar DataGuide incompatibility is reflected by the performance results for *CitiesMan* and *XMarkAuto* (figures 5.4 and 5.6, respectively). Here the optimization is practically irrelevant to the three CADGs, but clearly counter-productive for the DataGuide. However, since the *CitiesAuto* suite is more realistic as far as tree queries are concerned, this should not be taken as an argument against interval encoding on document nodes even for the DataGuide. Yet there are similar alternative techniques with an even greater optimization potential, which could be compatible with DataGuide (see section 5.4.4 for a short discussion).

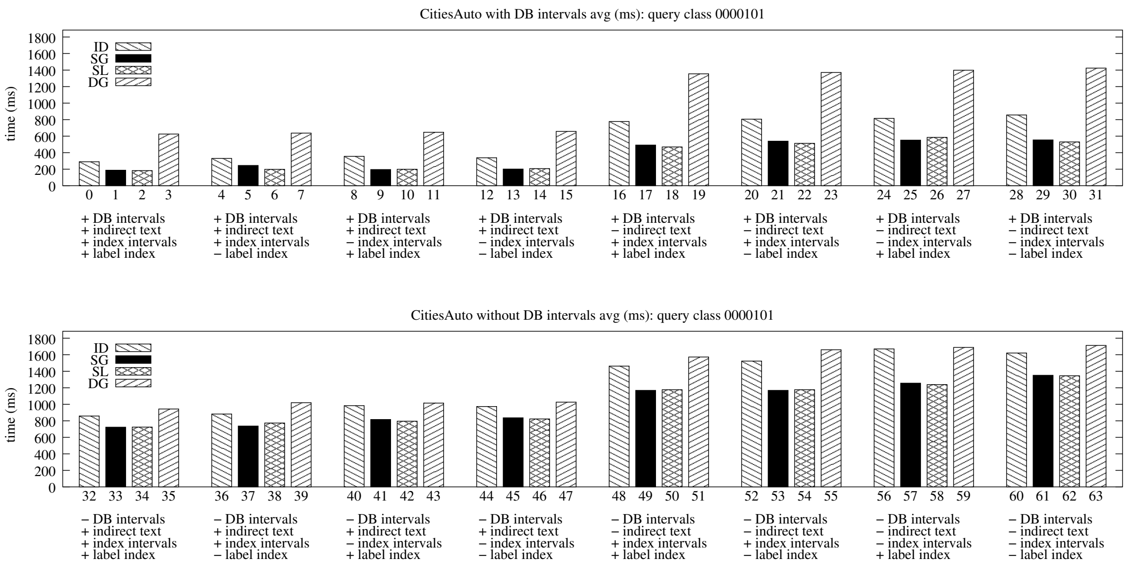


Figure 5.10: Performance gain of interval encoding on document nodes

5.4.3.3 Structural node identification of index nodes

The only optimization which seems to have virtually no effect at all is interval encoding used for index nodes. Section 3.3.3 points out that contributions to retrieval enhancement, if any, are most likely to be observed with huge index trees. Due to its sparse keyword distribution and poor retrieval performance for unselective queries (see section 5.4.4), the *NP* collection could not be used for extensive and systematic evaluation of this optimization technique. However, no salient effect has been observed for the other document collections in conjunction with the ID-Comparison CADG or the label index, except a slight improvement in combination with the latter for a single query class (see below). This does not necessarily mean that the hypotheses from section 3.3.3 are false. It seems more probable that the benefit of interval-encoded index node IDs, as a pure main-memory optimization, is just to weak to compensate for other prevalent factors, such as the navigational effort or the number of occurrence fetches during retrieval phase 2.

Index node intervals do not require much memory during the retrieval process or when storing the index persistently. Yet they do entail a certain (although relatively modest) implementation overhead and, most importantly, make the index structure more difficult to update incrementally. Together with the hardly promising performance results produced in the four test suites, these issues discourage the use of interval encoding for index nodes.

5.4.3.4 Label index

The last optimization technique from section 3.3, the label index, is responsible for subtle trends in the results only, far from producing such salient performance differences as e.g. the indexing of indirect keyword occurrences. Unlike interval encoding on index nodes, however, the label index causes no significant overhead or obstacle to incremental updates. Its benefit is best assessed by indirect statistical reasoning. For example, in the test suite *CitiesMan*, all index types need the label index in their configuration in order to have query classes with many soft structural edges among the three best-supported classes. Similar statements, only with restrictions for some tuning parameters, apply to the *CitiesAuto* and *XMarkAuto* suites. A single query class where a (still faint) effect of the label index can be observed directly is plotted in figure 5.11. Both the ID-Comparison CADG and the DataGuide follow a characteristic alternation pattern when comparing the groups of four boxes to each other.

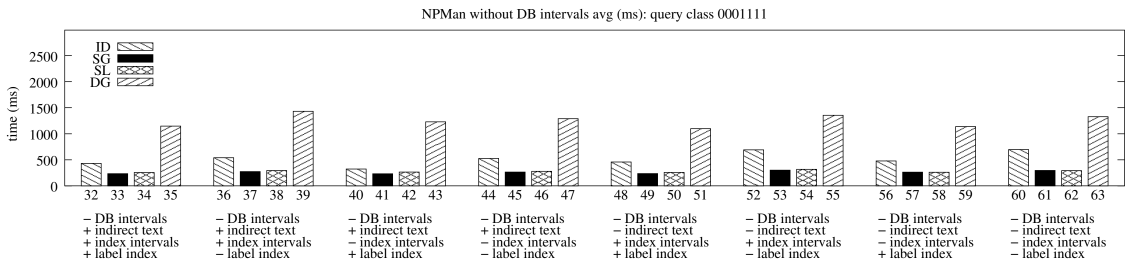


Figure 5.11: Performance gain of the label index

5.4.4 Remarks

Three issues have not yet been mentioned, or only touched briefly. The first concerns the fact that the benefit of content-awareness seems to depend much more on the query keywords' node selectivity than on their path selectivity. This contradicts the hypotheses formulated in chapter 3, which assume that path pruning based on indexed keywords should work best for keywords which occur below a limited number of label paths. Yet neither direct nor indirect analysis methods produced any clue as to the role of path selectivity. By contrast, there are characteristic patterns linked to the node selectivity of query keywords in the raw data, which are not part of the hypotheses. A more thourough analysis of the keywords employed in the query sets (especially the ones created automatically) might hint at some latent dependency between the keywords' node and path selectivity. However, this has not been explored in the course of this work.

A second issue regards the query classification proposed in section 5.2.3.1, and might well be related to the first point just discussed. For some query classes in figure 5.2, the performance gain of the Signature Look-Up CADG over the ordinary DataGuide varies considerably among the four test suites. While this phenomenon is not necessarily to be considered an artefact, it may in some cases be caused by a misclassification of query keywords or entire query trees. Again, a more detailed analysis of the query sets could clarify this issue.

Finally, the tests carried out during this experimental evaluation prove that the reconstruction of label/ID paths from retrieved keyword occurrences, as required for processing tree queries, needs further optimization. While the current query evaluation algorithm (see section 4.4) is conceptually reasonable, its implementation disregards the fact that for large document collections such as

NP, the stepwise reconstruction of the document paths leading to selected keyword occurrences causes too many database operations, especially when querying node-unselective keywords. This effect has caused considerable problems during query generation for the *NP* collection, and eventually lead to the manual query set devised for the *NPMan* suite. There are techniques allowing to minimize the number of disk accesses needed during path reconstruction, or to avoid them altogether (see section 6.7). Such optimizations may considerably improve the scalability of the evaluation algorithm. In this context, the benefit of interval encoding applied to document nodes should be revised, its functionality being provided by other structural node identification schemes which additionally support path reconstruction.

5.5 Storage requirements

Table 5.9 lists the storage requirements of the four index types, including all tables and the serialized navigational structure, for the three document collections. The maximal and minimal values in each field refer to indexing with and without interval-encoded document nodes, respectively. The statistics show that content-awareness entails a significant storage overhead, due to the combination of structure and content in a single content/annotation table (see section 3.2.1.1). While the ID-Comparison CADG and the Signature Generation CADG take up the same amount of memory on disk, the Signature Look-Up CADG's signature table further increases the storage requirements. For large document collections, however, this difference among the CADGs is not significant. The modest storage consumption of all indices for the biggest document collection, *NP*, is due to its sparse content (mostly a single keyword per document node). For content-rich collections such as *XMark*, CADGs become much larger than the data being indexed. Note, however, that unlike most relational index structures the approaches presented here are designed to replace the document collection. While in the test implementation the original document data are still needed for path reconstruction, this dependency can be avoided by using either certain structural node identification schemes (see section 3.3.2) or a secondary parent/child index, which is much smaller than the document table used so far.[32] Experiments show that for *Cities*, such a secondary index needs 1 MB on disk, for *XMark* 25 MB, and for *NP* 270 MB, i.e. about a third of the size of the original data.

doc. coll.	size in the database (MB)								
	doc. size	+ind. occ.				−ind. occ.			
		ID	SG	SL	DG	ID	SG	SL	DG
Cities	3	10-12	10-12	13-14	2-4	2-3	2-3	5-6	2
XMark	72	240-346	240-346	250-356	24-30	62-83	62-83	72-93	13-17
NP	604	103-134	103-134	119-150	32-51	22-30	22-30	37-45	15-22

ID: *ID-Comparison CADG*
SG: *Signature Generation CADG*
SL: *Signature Look-Up CADG*
DG: *DataGuide*
doc. coll.: *document collection*
doc. size: *size of all source documents together*
+ind. occ.: *with support for indirect keyword occurrences*
−ind. occ.: *without support for indirect keyword occurrences*

Table 5.9: Storage requirements

[32] Depending on the tree query language being used, there may be specific features other than path reconstruction which require access to the original document data, or else further secondary data structures. Most importantly, a query language may support the selection of a document node's entire textual content based on individual keywords. In such a case the original document content needs to be kept on disk, instead of being replaced by the index structure.

5.6 Recommendations for index tuning

There are several conclusions to be drawn from the preceding discussion, which are relevant for choosing the right index configuration. First, while the absolute evaluation time for a query certainly depends on the size of the index (and thus indirectly on the document collection), the relative performance gain of one index configuration over the other is much more influenced by the complexity of the query and by the tuning parameters applied to a given index structure. In other words, even though the sheer size of the data to be indexed and searched limits the absolute performance of a retrieval system, finding the appropriate index type and optimizations for a given retrieval scenario can considerably enhance its overall performance. Second, there are fewer relevant options to be considered than proposed in chapter 3. Among the different types of index compared in this work, the Signature Look-Up CADG is clearly the most efficient and the most flexible. One might argue that since three out of four test query sets contain slightly more mismatching than matching queries, the Signature Look-Up CADG is perhaps privileged in the tests. However it seems unlikely that with different query sets it could be beaten by the ID-Comparison CADG or the Signature Generation CADG. The combination of heuristic content representation and signature look-ups has been shown to be most effective even for *XMark*'s wealth of textual content.

Choosing the right optimization techniques from the ones introduced in section 3.3 is a little bit more complex. The easier part concerns the index node identification and the label index. Although designed for specific types of query, neither of the two contributes much to the overall performance, or even shows strong dependencies on specific query characteristics. Interval-encoded index nodes also have certain drawbacks, so that there is little interest in using them given their poor performance. The label index seems promising for very large index structures, although this has not been tested extensively. Its behaviour has been shown to depend largely on the query being evaluated. In the end, its benefits and risks are quite balanced. Unless a more sophisticated way is developed to determine when a look-up in the label index is worth while, it is probably best to save the extra effort and ignore it.

The other two optimizations, indexing of indirect keyword occurrences and interval encoding on document nodes, are much more efficient. The first technique almost constantly enhances retrieval performance and faces only modest costs at query time. However, the increased index size is the critical factor. As a rule of thumb, one may say that whenever the considerable storage and index creation overhead is acceptable, indexing of indirect keyword occurrences is worth while.[33] Interval-encoding document nodes is ambiguous: on the one hand, it clearly speeds up the evaluation of tree queries. On the other hand, when processing simpler queries the overhead tends to deteriorate the effect of other optimizations, such as content-aware navigation and occurrence fetching. The results from the *CitiesAuto* test suite suggest that despite its drawbacks, this optimization is recommended for real-world scenarios. As discussed earlier, however, there still is some optimization potential related to this issue, which proves to be a bottleneck in certain cases. The same is true for the path reconstruction problem discussed above. Solutions to these problems can be expected to have a considerable effect on the overall retrieval performance.

[33] This does not apply to less common retrieval scenarios where queries select mainly structure without content, of course.

Chapter 6

Related work

This chapter gives an overview of several earlier approaches to indexing Semi-Structured Data, all of which are related to the Content-Aware DataGuide to a greater or lesser extent. Clearly, the original DataGuide has served as a starting point and a basis to build upon. But there are also other DataGuide-based index structures, such as the IndexFabric (see section 6.2), the Signature File Hierarchy (see section 6.3), or the BUS index (see section 6.4) striving to enhance the integration of content and structure matching within a single index structure. Like the CADG, these indices are suitable for acyclic document collections only. Furthermore, two different approaches are covered here mainly for contrast: the Context Index, which has been designed to enhance structure matching, and the CIS Index, whose representation of document paths is fundamentally different from the DataGuide's. Finally, some work related to the index optimizations from section 3.3 is mentioned. All approaches covered in this chapter are only sketched briefly. For a more thourough examination, the reader is referred to [Wei02] where a detailed overview of these and other indexing techniques for Semi-Structured Data is given.

6.1 DataGuide

The original *DataGuide* [GW97a, GW97b, MAG$^+$97] is described in section 2.4, where also its main deficiency is discussed: a query's structural and textual selection criteria, i.e. the paths in the query tree and the keywords attached to them, are processed sequentially rather than in parallel. The first retrieval phase, where label paths are searched in the DataGuide, ignores the query keywords entirely. The same is true for the annotation fetching step in retrieval phase 2. Conversely, keyword occurrences are retrieved from the database regardless of the label paths they are reached by. The two selection criteria are only combined in a third phase, when the expensive database look-ups have already been accomplished. The sample query examined in section 2.4.3 illustrates that this separation of structure and content matching may entail many needless disk operations.

The Content-Aware DataGuide presented in chapter 3 remedies this defect by tightly integrating structure and content matching, which leads to considerable performance gains for a wide variety of queries (see chapter 5). The CADG's navigational structure is enriched with content information allowing to prune irrelevant paths in the index tree at query time. This saves both navigation effort in retrieval phase 1 and expensive disk accesses in phase 2. Besides, thanks to the intertwined content and structure matching in the early retrieval phases, there is no need for an explicit content/structure join any more, which is now carried out in the database on the level of index nodes rather than document nodes. Instead the query needs to be preprocessed in an additional retrieval phase 0 before navigation. Depending on the realization of the CADG, this may, but need not, involve a single database access. Extensive experimental evaluation suggests that disk-based approaches are superior to main-memory resident query preprocessing (see section 5.4.2.2). A more important, though probably not a critical aspect of the CADG is its increased storage consumption, compared to the DataGuide. Section 5.5 states that while for small document collections the relative storage overhead is considerable, although still manageable in terms of the absolute amount of memory needed, the CADG's storage requirements get much closer to the DataGuide's when indexing larger document collections. Here the overhead drops well below a quarter of the original size of the data.

6.2 IndexFabric

Another combined structure/content index based on the DataGuide is the *IndexFabric* [CSF+01, SCF+02, CS01], which is in some respect closely related to the Content-Aware DataGuide. The IndexFabric is best described as a DataGuide which has a specific kind of content index attached to each of its nodes: for any given index node, its annotations are stored in a *PATRICIA Trie*[34] built over all keywords occurring in document nodes referenced by this index node. The Trie's root is the index node itself. Accordingly, the IndexFabric can be viewed as indexing *occurrence paths* consisting of a sequence of labels in root-to-leaf direction, followed by the sequence of characters making up the keyword to be indexed. Likewise, a query path to be processed consists of a sequence of labels in root-to-leaf direction, followed by the sequence of characters making up the query keyword. When an index node with a matching path is reached during navigation, its annotations are not simply fetched from an annotation table and then intersected with a set of keyword occurrences to be looked up in a content table, as with the DataGuide. Instead the query keyword's characters are matched to the index node's PATRICIA Trie one by one, in a navigation procedure similar to the path matching performed before. If a query keyword occurs in some document node with the same label path as the index node being examined, the keyword matching in this index node's Trie succeeds, retrieving the matching document node as a hit. In this sense the IndexFabric supports content-aware occurrence fetching, just like the CADG. Both index structures perform the content/structure join on the level of index nodes instead of document nodes: as stated in section 3.1.2, both the query's structural and textual selection criteria are used simultaneously during occurrence fetching. This saves a dedicated retrieval phase for the content/structure join, as required by the DataGuide. Apart from the organization of their respective content index, the CADG and IndexFabric differ in two respects: while the CADG features content-aware navigation, unlike the IndexFabric, the latter comes with a paging strategy for efficient storage and retrieval of the possibly large index structure.

This similarity between the IndexFabric and the Content-Aware DataGuide can be pursued farther, back to section 3.2.1 where a *content-centric* variant of the latter is contrasted with the *structure-centric* one employed in the rest of this work. Applying the same duality to the IndexFabric, the index structure described in [CSF+01] and elsewhere is structure-centric in the sense that the content index, which could have been organized as a single PATRICIA Trie, is divided into index-node specific sub-Tries each covering only the keyword occurrences below its respective index node's label path. Every sub-Trie corresponds to one of the subtables attached to all index nodes in figure 3.3 on page 32. Analogously to the content-centric CADG shown in figure 3.1 on page 27, one could also imagine a content-centric IndexFabric consisting of multiple keyword-specific sub-DataGuides. It would look like a PATRICIA Trie covering all keywords in the document collection, with a keyword-specific DataGuide (like the ones shown in figure 3.1) attached to each node in the Trie. Interestingly enough, the content/structure duality is thus reflected, in the case of the IndexFabric, by the order of keywords and labels in an indexed occurrence path. While the occurrence paths indexed by the structure-centric IndexFabric consist of labels followed by keyword characters, as described above, a hypothetical content-centric IndexFabric would index occurrence paths with structural and textual query criteria in reverse order. The content-centric CADG is dropped as inefficient in section 3.2.1.4 because of its main deficiency, which is that keyword-specific sub-DataGuides need to be loaded from the disk at query time. By contrast, since the IndexFabric comes with a B-tree-like paging algorithm, its content-centric variant might

[34] For more information about the PATRICIA Tries and the Trie in general, see [Wei02] for a short overview, or the original papers [Mor68] and [Fre60], respectively.

perform reasonably well, and even better than the original IndexFabric when processing queries with path-selective keywords.

Besides the IndexFabric's paging strategy, which could also applied to other index structures including the Content-Aware DataGuide, there are other differences between the two approaches. Most importantly, the CADG supports content-aware navigation, i.e. the pruning of irrelevant paths in the index based on query keywords. Section 3.1.1 summarizes the benefits of this feature, whose effect is salient in the experimental results, too (see chapter 5). As a general concept for enhancing navigational index structures, content-aware navigation in either of the proposed realizations (based on signatures or index node IDs) could also be adapted to the IndexFabric, of course. The same is true for the optimization techniques discussed in section 3.3, most notably the indexing of indirect keywords occurrences which is proved highly effective by the experiments. In the case of the structure-centric IndexFabric, this would result in far bigger PATRICIA Tries attached to the nodes of its navigational structure. Like the CADG's content table, these content indices are stored on disk, such that the IndexFabric's paging strategy could help to make the optimization even more efficient than for the CADG. But even without support for indirect keyword occurrences, an empirical comparison of the two index structures seems promising.

6.3 Signature File Hierarchy

One of the two CADG realizations introduced in section 3.2.3, the Signature CADG, is closely related to the *Signature File Hierarchy* [CA98, CA99]. This index structure, which is also based on the DataGuide, features content-aware navigation (not occurrence fetching) using signatures like the Signature CADG, but with different data structures and techniques. Query node signatures are created and attached to the query tree as described in section 3.2.3.2. Yet a major difference between the two approaches lies in the content information added to the navigational structure. Every index node in the Signature File Hierarchy is associated with a *signature file* containing the node's annotations, along with signatures providing heuristic content information about each of the referenced document node. These document node signatures are created from all keywords in a document node, in much the same way as query node signatures are made from the query keywords' signatures. The signature file may be stored as a list of document node/signature pairs, or alternatively as a bit-labelled Trie, which makes it easier to search for signatures with specific bits set. Unlike the Signature CADG, there are no index node signatures in the Signature File Hierarchy.

The relevance check is carried out on the level of individual document nodes rather than collectively for all annotations of an index node, as with the Signature CADG. When an index node is reached during retrieval phase 1, is signature file is searched for document nodes with a signature matching the keywords attached to the query path being evaluated. The matching procedure is realized as a bitwise implication, as explained in section 3.2.3.2 for the Signature CADG. In case no matching entries exist in the signature file, the relevance check fails and the query is rejected as unsatisfiable. Otherwise, path matching continues. Annotation and keyword occurrence fetching is carried out separately on two tables during phase 2, for the Signature File Hierarchy just like for the DataGuide. Likewise, the joining of structural and textual hits is based on an intersection of document node sets which takes place in a dedicated retrieval phase 3. In other words, both navigation (although content-aware) and occurrence fetching with the Signature File Hierarchy stay on the level of document nodes rather than index nodes, and therefore face the same drawbacks as with the DataGuide.

The Signature CADG, by contrast, prunes paths during retrieval phase 1 using index node signatures, which simultaneously represent the textual content of all document nodes with a given label path (occurrence-fetching signature) or label path prefix (navigation signature). The occurrence fetching signature attached to any of its index nodes equals the bitwise disjunction of all signatures in the signature file of the corresponding Signature File Hierarchy node. Additionally, the relevance check is enhanced by the navigation signatures, which also contain keyword information about index nodes below the one being examined. Together these two signatures save the inspection of many signature files during navigation. Hence the Signature CADG's relevance check can be expected to be much faster than the Signature File Hierarchy's, whose signature files may even grow too big to be held in main memory at query time. The CADG requires at most a single database access during query preprocessing, but can also do without (see section 4.2.3). On the other hand, the Signature File Hierarchy's heuristic, keeping the content information of all document nodes apart, may be more precise than the Signature CADG's, both for navigation and occurrence fetching. However, this advantage may be annihilated by the higher fetching and join penalty of the Signature File Hierarchy.

6.4 BUS index

The *BUS index* [SJJ98, Shi01] is another DataGuide-based approach to indexing Semi-Structured Data. Besides the fact that the BUS index includes a ranking mechanism for keyword occurrences, which is not considered here, the main difference compared to the CADG lies in the respective content/annotation index used. The BUS index employs a content table which maps keywords to tuples containing not only the document node where the indexed keyword occurs, but also the referencing index node, its level in the navigational structure, and further information related to keyword ranking. There is no annotation table. In retrieval phase 2, when fetching the occurrences of a given keyword for an index node selected in the navigation phase, the index node information in the content table allows to discard occurrences which do not match the query path. In this sense, the BUS index performs a content/structure join at the level of index nodes rather than document nodes, like the CADG. Note, however, that in every content table entry, the information about the referencing index node is not part of the entry's key but rather of its value. As a consequence, it cannot serve as a selection criterion during occurrence fetching. In other words, structural query criteria are not used for content-aware occurrence fetching, as is the case for the CADG, but merely for filtering mismatching keyword occurrences after they have been fetched from the database. Therefore the BUS index needs a separate content/structure join phase, just like the DataGuide, in which all of a query keyword's occurrences must be examined to find out which of them are referenced by the right index node. Compare this to the CADG's content-aware fetching procedure which only retrieves occurrences with the right label path, integrating the content/structure join with the selection in the database.

Another disadvantage of the BUS index is that it cannot process pure structure queries. Due again to the design of its content index, there is no efficient way to get hold of a given index node's annotations regardless of their textual content. When processing queries without keywords, the DataGuide relies on its annotation table to retrieve the matching index nodes' annotations. The CADG looks them up in its content/annotation table just like for any other query, simply omitting textual selection criteria. The BUS index, by contrast, cannot answer the query, lacking a mapping from index nodes to sets of document nodes. The only workaround, an exhaustive search in the entire content table (without the use of a secondary index), is clearly unfeasible at query time.

The query evaluation procedure described in [SJJ98] retrieves document nodes which contain the query keywords either directly or indirectly (see section 2.3 for a discussion of direct vs. indirect textual containment). In terms of the query formalism used in this work, all textual query nodes are assumed to be implicitly soft-edged. Since the BUS index does not index indirect keyword occurrences (see section 3.3.1), they are reconstructed from direct occurrences as sketched in section 2.3. To avoid frequent database look-ups, document nodes are identified according to the Virtual Nodes scheme (see section 6.7 below) which allows for path reconstruction in main memory. Chapter 5 provides experimental evidence that path reconstruction involving database look-ups is a major bottleneck when processing non-trivial tree queries with unselective keywords. Hence optimizations based on structural node identification of document nodes can be expected to contribute much to the overall performance of a retrieval system. The approach taken by the BUS index is also applicable to other index structures, including the CADG and the ordinary DataGuide.

6.5 CIS Index

All indexing approaches reviewed so far have in common that they represent document paths as compositional objects which can be linked to form tree structures, and navigated during query evaluation. Accordingly, [Wei02] classifies them as *navigational*, in contrast to *elementary* and *path look-up* indices. In terms of this classification, the *CIS Index* [Meu00], which is introduced in section 4.1.2, belongs to the third kind of index structure. Unlike the CADG and the DataGuide, it stores document paths as atomic units, which remain separated even if they share the same path prefix. Navigational indices, by contrast, represent each document path only once, whether is ends in a leaf or in an inner node. Path fragments common to many document paths, such as the prefix /book/chapter in figure 2.2 *(b)* (see page 17), are not duplicated. Avoiding redundancy in this way reduces the storage overhead considerably, compared to the CIS Index. Experiments show that for large document collections (more than 500 MB), the storage required by the CIS Index exceeds both the DataGuide's and the CADG's size by an order of magnitude. For small document collections (less than 5 MB), the difference still amounts to a factor 4. When indexing indirect keyword occurrences with the CADG (see section 3.3.1), the CIS Index is twice as large as the navigational index whose size increases considerably with this optimization, but again nearly ten times larger than the DataGuide.

Section 5.4.4 points out that the path reconstruction needed for joining multiple query paths is a bottleneck in the evaluation of tree queries as described in section 2.3. This critical phase profits most from the redundancy inherent to the CIS Index. As a matter of fact, the label/ID paths it stores for every keyword can be considered the precomputed result of path reconstruction. In this respect the CIS Index may be more efficient than the navigational index structures evaluated in chapter 5, although no experimental results are available. Note, however, that there are more sophisticated techniques to address the path reconstruction problem while avoiding excessive redundancy. An example is provided by the BUS index (see section 6.4), where the Virtual Nodes encoding scheme allows for in-memory path reconstruction (see the next section). Hence it seems more promising to apply similar optimizations to the CADG rather than trying to minimize the storage requirements of the CIS Index.

6.6 Context Index

An approach to enhancing retrieval with heuristic structural rather than textual information is the *Context Index* [SM99, MS99a]. Conceptually similar to an inverted list such as the DataGuide's content index (see section 2.4.1), the Context Index additionally uses *path signatures* to represent the label paths leading to indexed keyword occurrences. Attached to the document node IDs stored in the index table, they allow to discard some of the occurrences whose label paths do not match the structural query criteria, without examining the label path in detail. This optimization can be regarded as complementary to the CADG's content-awareness during navigation and occurrence fetching. Note, however, that while in the latter case content-awareness actually saves database accesses, the Context Index only enhances path processing after the fetch. Structural query criteria are not used for the selection in the database (which would require the path signatures to be part of a path table entry's key rather than its value). Hence the context information serves a-posteriori filtering, rather than a-priori selection as with the CADG.

6.7 Optimization techniques

Section 3.3 presents four optimization techniques for use with navigational index structures, which are evaluated experimentally in chapter 5. Here some related approaches are listed with a brief comment on each of them. For a more detailed discussion, refer to [Wei02], or else to the original papers.

Indexing of indirect keyword occurrences (see section 3.3.1) is mentioned in [LYYB96] as one of several schemes for inverted indices, termed *ANWR* (for *A*ll *N*odes *W*ith *R*eplication). The paper also points out that this technique causes considerable redundancy, as the experimental evaluation in section 5.4.3.1 confirms.

Structural node identification of document or index nodes (see section 3.3.2 and 3.3.3, respectively) can be accomplished using different numbering schemes. Unlike the one used in this work, *interval encoding* [LM01], the *Virtual Nodes* scheme [LYYB96] is based on a breadth-first rather than a depth-first traversal of the document tree. While it is easy to apply either kind of numbering scheme to index nodes, creating document node IDs in a breadth-first traversal may be more difficult, e.g. when using a depth-first parser (which is straightforward and most common for XML documents) or when processing a stream source. On the other hand, the Virtual Nodes scheme allows to compute parent IDs from given child IDs without accessing the database, which is most valuable for enhancing path reconstruction (see section 5.4.4). The drawback is that many IDs need to be reserved for non-existing ("virtual") nodes, which increases the size of the individual identifiers and thus the overall storage requirements. Both approaches to structural node identification are recommended mainly for static document collections, although providing limited support for database updates.

As stated in section 3.3.4, the label index, although designed to enhance path matching in the index tree, has a restricted filtering function. When a label is looked up which does not exist in the document collection, the label index saves needless navigation of possibly large parts of the index tree, depending on the actual structure of the query. In this respect it resembles the Context Index (see section 6.6), which serves to identify label mismatches, too. However, the use of the label index for this task is far more restricted because the indexed labels are not linked to keyword occurrences in any way. Only the presence or absence of labels can be deduced from a look-up in the label index, and not in a path-specific manner but globally for the whole document collection.

Chapter 7

Conclusion

7.1 Results

In the previous chapters, a novel approach to indexing Semi-Structured Data, the *Content-Aware DataGuide* (CADG), has been first motivated, then explained from a conceptual point of view, afterwards evaluated empirically with several variations and optimizations taken into account, and finally contrasted with related index structures and techniques. Section 5.6 summarizes the analysis of the experimental results. Among the salient conclusions to be drawn is the fact that the CADG outperforms the DataGuide in practically all cases. The relative performance gain, although depending on the document collection being queried, is often near to 50% and sometimes even much greater. This proves that the CADG's major contributions, *content-aware navigation* and *content-aware occurrence fetching*, substantially enhance a system's retrieval performance whenever content information is queried.

Among the three different realizations of the CADG, the heuristic *Signature Look-Up CADG* proves most efficient and flexible. It prevails most clearly when processing textually mismatching queries. Interestingly enough, looking up keyword signatures in the database at query time is more efficient than creating them on the fly. The exact ID-Comparison CADG cannot compete with the two Signature CADGs when a query entails much navigation, due to its increased content-awareness overhead. The three CADGs, above all the Signature Look-Up CADG, require significantly more storage than the original DataGuide, although even for larger document collections the space consumption is still manageable.

Other interesting observations arise from the assessment of four optimization techniques which have been evaluated with all index structures. While indexing of indirect keyword occurrences is highly effective, but fairly expensive in terms of storage, interval encoding of index nodes and a secondary label index turn out to be mostly useless. Applied to document nodes, however, interval encoding clearly shows a certain potential for non-trivial tree queries. However, there are also some side effects which eventually slow down the processing of simpler queries. Together with the path reconstruction problem related to the current query evaluation algorithm, this suggests that there may still be more efficient optimization techniques to be discovered.

Methodically, the analysis of the experimental results has revealed subtle differences between hand-crafted and automatically generated query sets. In the end, it is obvious that a careful evaluation should not be based on either kind of query creation alone.

7.2 Future work

Some of the remaining research issues have already been hinted at above. For instance, the query evaluation algorithm should be optimized to avoid the sometimes huge amount of disk operations needed for path reconstruction. This would not only improve the retrieval system's performance but also produce more exact test results. As pointed out before, some of the present experimental results suffer from artefacts caused by this deficiency in the query evaluation algorithm. Other possible enhancements include a look-up table for the ID-Comparison CADG, similar to the Signature Look-Up CADG's signature table, or heuristics for fine-tuning the use of a label index based on query and index properties.

A more important issue is related to index updates. Currently no provisions have been made to update the index structures incrementally, i.e. to modify only certain parts of them instead of just reindexing the whole document collection. It should be examined in how far the incremental update mechanism proposed for the DataGuide is applicable to the CADG and possible optimizations. In the long run, one might also explore the question whether the CADG can be adapted to DAG-shaped documents or even arbitrary graph databases. However, this would require much more thorough investigation.

As far as the experimental evaluation is concerned, improvements can be expected from a more precise query classification. First of all the automatically generated query sets should be checked for misclassified queries. An additional query characteristic, distinguishing structural from textual mismatches, would also allow for a more exact analysis. A new query generator operating on the index tree rather than the DTD of a document collection would drastically reduce the number of undesired mismatches, and thus make it much easier to create new query sets. Alternatively, one may choose to analyze the existing data more thoroughly first. In any case, the interesting document collection *NP* has not yet been fully exploited.

Appendix A

Experimental results

The following eight tables list the experimental results obtained from the four test suites *CitiesMan*, *CitiesAuto*, *XMarkAuto*, and *NPMan*, which are described in section 5.2. Each table column represents an index configuration. The tables contain all queries which have been taken into account for the performance analysis in chapter 5. For each query, its class and ID are provided in the first two columns of every table. The performance values represent an average over three (in the case of *CitiesMan* five) iterations of the same query, in milliseconds. A detailed description of the experimental setting can be found in section 5.3 on page 67.

Table A.1: *CitiesMan* absolute (ms): average over all iterations of a query with DB intervals

class	ID	+DB intervals																															
		+indirect keyword occurrences																−indirect keyword occurrences															
		+index intervals								−index intervals								+index intervals								−index intervals							
		+label index				−label index				+label index				−label index				+label index				−label index				+label index				−label index			
		ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG
		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0000000	0a	5805	6932	6053	10701	5664	6896	5927	10683	5709	6615	5967	10686	5672	6574	5873	10568	19569	25219	22267	31884	19528	24936	21937	31649	19205	24906	21770	31446	18818	24718	21466	31299
0000000	0b	4821	6220	5455	8390	4802	6121	5414	8347	4786	9769	5325	8327	4781	5885	5323	8264	11534	14947	13213	18998	11417	14847	12979	18924	11380	14743	12859	18720	11181	14679	12763	18675
0000001	1	1458	1036	930	3261	1442	1017	922	3250	1477	1523	941	3257	1482	998	922	3231	2457	2379	2163	4824	2456	2348	2116	4824	2479	2349	2127	4769	2431	2357	2103	4770
0000010	2	2917	3431	3071	5555	2874	3412	2968	5521	2945	4936	2992	5498	2866	3269	2935	5406	8270	10617	9337	13912	8233	10452	9190	13799	8208	10473	9131	13683	8036	10409	9008	13618
0000011	3a	1273	739	657	2969	1274	717	665	2956	1297	1203	659	2986	1266	710	655	2940	1327	823	758	3123	1359	800	743	3082	1452	816	742	3086	1341	812	736	3079
0000011	3b	570	375	327	1352	593	362	322	1380	593	630	338	1357	588	348	326	1354	738	478	430	1867	753	471	419	1879	859	492	413	1860	770	457	422	1845
0000100	4a	3758	4299	3790	7105	3812	4278	3697	7057	3827	4847	3729	7031	3711	4054	3631	7001	3659	4037	3580	6967	3618	4026	3499	6932	3827	4021	3550	6905	3638	3988	3503	6882
0000100	4b	251	284	276	511	291	304	256	531	282	287	255	505	282	267	278	522	278	269	253	521	276	287	279	521	277	268	254	519	279	291	279	541
0000101	5a	1302	849	769	2999	1253	839	765	3004	1293	831	760	2992	1298	847	771	2995	1228	776	728	2974	1202	773	701	2957	1267	799	720	2962	1237	782	744	2937
0000101	5b	110	126	111	286	134	140	118	299	144	131	106	262	129	134	116	281	106	107	103	275	157	133	119	295	112	106	102	279	162	134	113	293
0000110	6a	2100	2332	2025	4011	2075	2314	2017	3990	2087	2234	2020	3960	2065	2212	1992	3949	2027	2186	1941	3925	2034	2167	1896	3916	2465	2175	1893	3866	2005	2157	1880	3834
0000110	6b	1410	1958	1706	2301	1412	1901	1655	2297	1385	1859	1605	2283	1405	1830	1640	2265	1370	1834	1579	2234	1377	1842	1582	2243	1652	1774	1580	2222	1351	1782	1559	2242
0000111	7a	1213	653	607	2903	1186	648	591	2866	1233	657	595	2892	1188	670	606	2887	1165	630	555	2860	1145	609	591	2839	1229	633	582	2836	1145	630	563	2841
0000111	7b	119	75	70	305	140	105	93	341	104	74	81	299	161	107	101	309	102	88	69	282	122	85	83	341	119	72	67	324	122	83	78	322
0001000	8	279	365	324	565	319	385	326	622	271	366	342	548	282	365	322	574	773	945	846	1561	800	955	853	1582	792	931	834	1559	787	960	856	1563
0001001	9	16	20	20	40	53	39	38	90	16	20	19	54	49	52	56	102	174	206	185	412	208	203	187	442	232	200	184	414	206	233	199	438
0001010	10	296	432	364	518	330	432	365	531	306	410	345	521	322	396	357	521	729	948	827	1255	740	966	842	1255	796	932	824	1238	744	933	808	1253
0001011	11	27	28	15	42	35	28	44	80	13	16	15	43	38	25	43	78	149	94	88	539	145	104	110	540	120	93	87	514	139	121	132	535
0001100	12	180	216	195	360	213	240	231	407	180	224	188	365	193	240	207	391	168	220	188	322	226	239	220	389	189	208	216	358	210	234	199	378
0001101	13	28	37	35	56	81	56	72	91	30	36	38	56	79	54	75	87	27	35	34	73	65	72	69	93	28	36	33	72	61	53	72	109
0001110	14	770	1036	917	1247	792	1035	892	1227	741	1009	883	1198	772	997	870	1210	745	979	875	1179	767	998	877	1174	740	968	841	1174	745	973	841	1172
0001111	15	24	32	30	94	67	56	41	113	24	30	29	80	46	40	37	132	39	31	29	93	62	41	37	114	24	29	46	95	45	37	37	111
0010000	16	15	16	15	108	13	16	17	107	13	15	15	123	12	15	19	107	8128	9980	8722	14026	7942	9889	8667	13948	7902	9871	8598	13861	7793	9767	8467	13758
0010001	17	41	28	21	24	12	15	16	24	13	16	15	24	13	15	16	24	1663	1400	1250	3668	1651	1409	1232	3670	1694	1401	1243	3645	1644	1419	1231	3598
0010010	18	6	7	8	54	6	7	8	36	6	21	8	36	6	6	8	35	3345	4012	3547	6132	3300	3988	3496	6077	3302	3977	3467	6031	3271	3947	3424	5990
0010011	19	8	10	10	22	8	9	11	12	9	10	12	10	8	9	10	10	1158	645	594	2881	1129	667	605	2865	1241	659	595	2834	1174	660	592	2844
0010100	20	2584	3160	2815	4212	2550	3101	2784	4173	2558	3074	2730	4138	2568	3066	2786	4163	2496	2969	2664	4117	2521	2997	2663	4091	2598	3015	2640	4097	2617	2992	2602	4058

continued on next page

Table A.1: *CitiesMan* absolute (ms): average over all iterations of a query with DB intervals
(continued from previous page)

class	ID	+DB intervals																															
		+indirect keyword occurrences																−indirect keyword occurrences															
		+index intervals								−index intervals								+index intervals								−index intervals							
		+label index				−label index				+label index				−label index				+label index				−label index				+label index				−label index			
		ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG
		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0010101	21	393	363	334	841	389	360	332	857	394	367	327	837	393	372	335	833	362	327	316	852	354	344	304	846	372	351	334	844	379	345	310	831
0010110	22	1441	1732	1532	2444	1423	1731	1492	2412	1413	1702	1512	2393	1409	1675	1502	2390	1381	1643	1478	2379	1401	1645	1469	2387	1399	1629	1447	2379	1379	1646	1435	2369
0010111	23	286	168	140	707	300	164	152	689	281	151	152	652	283	166	144	706	294	145	131	666	279	151	132	680	279	147	135	686	280	164	147	703
0011000	24	13	15	15	146	12	15	15	112	13	16	15	122	12	15	16	124	7961	9945	8721	14057	7952	9898	8650	13939	7844	9848	8612	13858	7776	9759	8457	13769
0011001	25	34	21	20	24	13	15	15	36	12	31	15	24	12	16	16	24	1650	1384	1261	3681	1671	1414	1230	3656	1687	1426	1249	3658	1653	1384	1244	3662
0011010	26	5	7	7	36	6	7	7	53	6	7	8	53	6	7	8	69	3331	4023	3523	6134	3306	3989	3484	6060	3295	3966	3480	6040	3256	3947	3410	5987
0011011	27	8	10	11	11	8	9	11	11	8	10	10	11	8	9	11	10	1129	669	594	2866	1165	647	599	2840	1191	666	602	2863	1175	646	575	2871
0011100	28	227	295	252	449	227	294	264	417	223	275	255	415	208	290	245	410	220	281	261	382	221	287	255	409	220	275	271	429	203	283	237	406
0011101	29	37	51	83	88	38	49	47	71	38	47	45	86	43	66	46	93	37	46	43	70	37	47	44	95	37	62	43	87	37	63	58	68
0011110	30	738	1052	895	1180	750	1001	866	1189	745	952	856	1160	771	966	847	1149	724	938	832	1163	737	966	851	1142	708	926	824	1129	699	938	825	1146
0011111	31	30	38	35	101	29	38	35	101	29	36	35	100	28	37	35	84	28	37	35	115	29	36	46	81	30	37	46	99	45	37	33	99
0100000	32a	327	182	179	866	323	182	198	862	300	219	198	880	300	217	178	853	4027	5214	4569	6658	3998	5160	4542	6613	3926	5134	4508	6573	3913	5124	4422	6490
0100000	32b	13	14	15	66	17	20	17	51	14	16	26	66	17	18	18	49	156	168	149	354	158	189	153	357	169	164	149	369	158	167	157	370
0100001	33a	242	144	154	589	238	149	148	577	258	143	138	601	241	144	149	584	316	266	252	738	333	266	246	743	318	267	246	729	332	274	249	724
0100001	33b	25	17	18	81	33	19	20	84	26	19	17	100	26	18	19	86	25	17	17	63	26	19	19	81	24	18	16	85	32	19	18	80
0100010	34a	265	167	143	717	282	166	140	703	296	149	143	719	292	150	152	669	2303	2914	2534	3927	2279	2884	2519	3877	2246	2848	2500	3869	2230	2838	2461	3842
0100010	34b	20	16	18	46	23	19	17	71	21	16	19	46	21	18	20	47	19	15	16	40	25	21	21	41	20	16	16	55	23	41	19	56
0100011	35a	248	141	128	530	248	129	128	556	250	125	114	528	231	130	130	533	235	127	116	576	250	127	115	555	256	138	119	527	250	152	118	548
0100011	35b	11	9	10	39	14	12	12	41	12	9	9	43	14	12	12	62	54	27	24	176	57	31	31	179	55	26	25	187	58	32	29	179
0100100	36a	822	882	924	968	791	830	839	940	895	840	824	957	791	803	794	912	816	834	797	978	813	820	775	947	814	834	834	903	764	874	793	941
0100100	36b	258	187	193	624	273	183	166	672	280	171	192	630	268	201	164	646	264	163	139	665	259	149	143	646	267	151	140	698	243	153	139	679
0100100	36c	65	58	59	52	89	56	52	90	51	54	48	53	74	57	54	79	77	53	60	75	74	69	93	100	53	69	50	53	92	56	94	75
0100100	36d	121	159	158	219	124	161	159	204	119	168	143	199	137	169	143	215	134	164	146	194	124	152	138	233	116	149	146	215	118	156	136	235
0100101	37a	227	139	126	529	231	138	125	528	247	138	133	512	229	136	141	525	232	128	134	554	233	143	117	590	242	152	120	570	219	131	118	583
0100101	37b	40	18	18	62	25	19	18	61	25	18	17	60	26	34	18	62	23	16	17	80	25	18	31	84	24	17	16	78	25	19	22	64
0100110	38a	249	154	132	567	248	152	143	580	239	140	129	580	256	140	128	563	227	144	121	633	225	137	119	629	228	136	140	609	247	149	126	631
0100110	38b	19	15	17	37	20	19	19	64	19	14	16	38	23	19	41	54	18	15	15	56	21	19	19	60	19	15	17	39	22	17	18	58
0100111	39a	225	139	110	509	217	137	110	508	247	122	112	508	224	120	127	507	234	133	106	523	227	115	105	514	222	119	108	582	218	119	106	519

continued on next page

Table A.1: *CitiesMan* absolute (ms): average over all iterations of a query with DB intervals
(continued from previous page)

class	ID	+DB intervals																															
		+indirect keyword occurrences																−indirect keyword occurrences															
		+index intervals								−index intervals								+index intervals								−index intervals							
		+label index				−label index				+label index				−label index				+label index				−label index				+label index				−label index			
		ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG
		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0100111	39b	27	8	9	37	14	12	13	60	11	8	9	37	15	11	12	40	11	8	8	37	14	12	12	78	11	8	8	41	14	12	13	41
0101000	40	2	2	3	18	9	8	9	24	2	2	3	33	7	7	9	30	41	40	36	82	48	45	57	93	41	42	36	80	58	45	41	125
0101001	41	2	2	3	2	7	9	9	9	1	2	3	2	7	8	8	8	59	21	20	102	61	27	32	91	56	21	20	85	61	26	38	96
0101010	42	2	2	3	9	8	8	9	18	2	2	3	10	7	8	8	14	392	528	464	681	402	528	460	684	385	520	454	727	392	539	459	696
0101011	43	2	2	3	2	7	8	9	7	2	2	3	2	7	7	9	7	19	16	17	120	25	22	22	125	35	17	16	132	25	22	21	125
0101100	44a	23	30	27	38	29	37	32	38	22	45	25	32	26	34	31	37	21	28	25	31	27	34	31	37	22	28	26	32	42	34	30	37
0101100	44b	14	19	17	30	21	25	39	58	14	18	17	28	40	23	23	35	14	18	17	27	20	24	24	59	13	33	17	45	20	22	28	32
0101101	45	4	6	6	6	10	11	12	12	4	5	6	6	10	12	12	11	4	5	6	6	10	12	12	12	5	6	6	30	10	11	12	11
0101110	46	304	388	338	440	290	392	329	459	282	371	332	434	286	373	336	461	279	365	323	457	281	368	335	436	283	389	347	420	261	364	320	450
0101111	47	3	4	5	4	9	11	11	11	3	4	4	4	9	9	10	10	3	3	5	4	9	9	10	12	3	4	5	5	8	9	9	10
0110000	48	11	21	26	34	12	15	15	38	11	14	14	33	11	15	14	34	2060	2636	2320	3532	2058	2635	2291	3498	2058	2619	2266	3488	2015	2598	2231	3465
0110001	49	9	11	11	13	8	12	11	13	8	11	11	13	8	11	11	12	277	207	177	591	275	208	181	603	266	194	208	583	260	213	176	585
0110010	50	3	4	4	11	3	4	5	13	3	3	5	14	3	4	4	12	1225	1542	1343	2194	1209	1493	1307	2159	1207	1467	1301	2132	1193	1474	1267	2145
0110011	51	3	4	5	4	3	4	5	4	3	30	5	17	3	4	5	4	215	119	108	515	212	120	108	507	241	145	111	527	231	122	118	511
0110100	52a	15181	20474	17977	22871	15076	20278	17762	22722	15033	19392	17670	22468	14913	19409	17403	22459	14935	19189	17224	22309	14784	19169	17055	22112	14626	18949	16804	21970	14541	18962	16735	21867
0110100	52b	1502	2007	1760	2433	1502	1972	1762	2402	1522	1924	1731	2371	1480	1893	1711	2363	1498	1922	1664	2346	1478	1881	1669	2311	1471	1874	1657	2324	1462	1826	1649	2318
0110101	53	156	120	97	318	140	107	93	306	164	103	97	318	141	123	94	298	144	102	103	318	136	100	90	325	140	105	90	295	138	113	90	310
0110110	54	955	1287	1131	1576	967	1254	1119	1593	929	1243	1128	1537	956	1235	1098	1562	944	1172	1066	1519	925	1183	1068	1532	922	1177	1049	1538	910	1189	1038	1508
0110111	55	61	59	55	209	76	59	53	205	82	59	55	190	62	59	54	223	59	56	53	217	58	61	51	222	62	58	53	215	60	57	52	220
0111000	56	12	16	14	38	12	15	14	42	11	15	14	35	13	14	14	54	2084	2645	2346	3585	2066	2629	2302	3573	2049	2623	2284	3572	2004	2602	2261	3545
0111001	57	8	11	16	13	9	11	26	13	8	11	11	13	21	11	11	13	285	193	180	665	264	228	177	663	287	230	180	629	284	194	181	644
0111010	58	3	4	5	14	3	5	5	14	3	4	4	13	3	5	5	14	1231	1483	1320	2228	1235	1494	1311	2214	1207	1493	1288	2232	1199	1486	1294	2219
0111011	59	3	4	6	4	3	4	5	4	3	3	5	5	3	4	4	4	217	135	111	567	235	126	108	573	237	124	112	571	219	135	112	549
0111100	60a	66	104	76	97	63	87	76	96	64	85	74	96	62	88	73	111	76	82	85	95	62	98	73	94	62	97	87	93	60	82	72	93
0111100	60b	47	66	58	88	46	65	56	98	59	63	57	84	46	62	56	83	45	60	55	81	45	61	67	81	56	66	54	97	45	60	69	96
0111101	61	15	21	35	35	15	21	20	23	15	21	19	35	16	20	20	24	15	24	20	23	15	19	18	23	15	20	19	23	16	35	18	23
0111110	62	432	599	536	678	429	600	514	688	433	588	513	666	424	560	523	679	423	560	500	676	420	553	495	670	417	552	488	644	429	548	481	645

continued on next page

Table A.1: *CitiesMan* absolute (ms): average over all iterations of a query with DB intervals
(continued from previous page)

class	ID	+DB intervals																															
		+indirect keyword occurrences																−indirect keyword occurrences															
		+index intervals								−index intervals								+index intervals								−index intervals							
		+label index				−label index				+label index				−label index				+label index				−label index				+label index				−label index			
		ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG
		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0111111	63	7	9	10	11	7	15	15	11	7	10	9	11	7	10	10	10	7	10	9	10	7	10	10	10	7	10	22	10	7	10	9	10
1000000	m0a	235	130	2	565	249	138	2	558	240	146	2	552	249	143	2	550	233	129	2	567	226	141	2	533	247	128	2	549	228	143	2	551
1000000	m0b	246	147	2	563	258	131	2	547	251	129	2	573	247	129	2	547	229	126	2	555	251	127	2	565	265	128	2	568	244	137	2	598
1000001	m1	229	129	2	586	239	147	2	584	251	204	2	565	264	127	2	551	234	142	2	563	234	139	2	547	249	150	2	562	254	142	2	550
1000010	m2	263	132	2	575	240	143	3	583	250	230	2	581	247	125	3	574	244	121	3	565	227	137	2	552	250	146	2	567	245	138	2	560
1000011	m3a	246	122	2	599	258	122	2	554	252	206	2	571	248	121	2	564	246	152	2	571	229	136	2	563	254	136	2	552	247	132	2	544
1000011	m3b	230	126	2	585	233	122	2	550	241	205	2	581	281	139	2	584	231	119	1	566	245	121	2	564	310	121	2	569	250	120	2	551
1000100	m4a	255	129	2	566	241	128	2	568	249	239	2	564	226	140	2	549	235	123	3	562	216	122	2	542	283	124	2	563	235	123	2	557
1000100	m4b	25	17	1	61	26	20	2	82	25	18	2	61	26	48	2	78	22	17	2	90	26	17	2	63	23	19	2	79	42	23	2	63
1000101	m5a	224	125	2	569	222	146	2	566	245	123	2	593	247	139	2	580	217	120	2	562	216	119	2	560	226	140	2	562	221	134	2	556
1000101	m5b	23	14	2	79	25	15	2	62	30	14	2	61	26	14	2	95	23	13	2	78	25	14	2	61	25	13	2	77	25	14	2	88
1000110	m6a	255	147	2	567	253	137	1	566	230	138	2	582	241	125	2	580	264	130	1	593	216	118	2	563	264	134	2	568	218	132	2	543
1000110	m6b	11	1	2	28	14	2	2	32	12	0	2	46	14	3	2	48	11	1	2	46	14	2	1	35	14	0	2	29	14	2	1	32
1000111	m7a	261	134	2	583	236	135	3	592	266	151	1	568	258	124	3	575	220	131	2	564	239	129	2	573	296	116	2	556	221	133	2	561
1000111	m7b	24	13	2	80	42	14	2	63	25	12	2	79	26	14	2	95	23	12	2	61	25	12	2	63	24	12	2	61	25	14	2	77
1001000	m8	2	2	3	2	10	6	2	9	2	2	2	2	9	5	2	26	21	15	2	180	25	18	2	171	18	16	2	164	40	18	2	169
1001001	m9	2	2	2	2	9	7	2	10	2	2	3	2	9	4	1	8	35	3	2	31	26	6	2	39	20	3	2	31	43	8	2	53
1001010	m10	2	2	3	2	8	5	2	9	2	2	2	2	9	6	2	9	40	19	19	99	50	23	2	85	30	37	2	96	29	23	2	105
1001011	m11	2	2	2	3	9	5	2	9	2	2	2	2	9	5	2	9	18	15	2	179	24	18	2	186	17	16	2	167	23	18	2	170
1001100	m12	2	3	3	2	10	5	2	11	2	2	2	2	8	5	2	10	2	2	2	2	9	38	2	9	2	2	3	2	25	5	2	9
1001101	m13	2	1	2	2	10	3	2	21	2	1	2	2	8	3	14	8	3	0	2	2	9	4	2	8	2	2	2	3	9	3	2	9
1001110	m14	2	1	2	3	9	3	3	9	2	1	2	2	9	4	2	9	2	1	1	2	9	3	2	9	2	1	2	2	9	6	2	9
1001111	m15	2	8	2	2	9	5	2	10	2	2	2	2	9	5	2	9	2	2	2	2	9	5	2	10	3	3	2	2	8	5	2	21
1010000	m16	2	2	2	2	1	2	2	2	2	3	2	2	2	2	2	2	213	131	2	557	211	156	2	577	233	123	3	556	229	119	2	554
1010001	m17	2	2	2	2	2	2	2	2	2	2	2	2	1	3	13	2	226	116	2	574	230	121	2	553	233	120	2	567	230	116	2	553
1010010	m18	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	210	119	2	570	225	130	1	552	234	129	1	568	246	132	2	583
1010011	m19	2	2	22	2	1	2	2	2	1	2	2	2	2	2	2	2	215	117	2	569	235	112	2	548	222	114	2	534	229	114	2	537

continued on next page

Table A.1: *CitiesMan* absolute (ms): average over all iterations of a query with DB intervals
(continued from previous page)

class	ID	+DB intervals																															
		+indirect keyword occurrences																−indirect keyword occurrences															
		+index intervals								−index intervals								+index intervals								−index intervals							
		+label index				−label index				+label index				−label index				+label index				−label index				+label index				−label index			
		ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG
		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
1010100	m20	61	51	2	194	73	49	2	208	58	55	2	191	73	55	2	203	56	50	2	206	54	50	2	206	57	67	2	190	56	49	2	205
1010101	m21	117	44	2	253	115	45	2	253	105	60	2	252	102	46	2	236	97	57	2	235	114	43	2	250	116	44	2	235	125	44	2	245
1010110	m22	56	53	3	207	55	51	2	207	57	52	3	208	71	51	2	200	54	50	2	191	55	67	1	205	57	51	3	190	67	50	2	202
1010111	m23	58	50	2	193	56	48	2	203	57	49	2	192	57	50	2	214	53	48	2	191	56	47	2	173	56	48	2	205	56	47	2	189
1011000	m24	2	2	2	2	1	2	2	2	2	2	2	2	2	2	2	2	230	118	2	560	209	134	3	572	248	119	3	556	219	125	2	550
1011001	m25	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	217	115	2	554	226	117	2	568	235	117	3	555	214	152	2	547
1011010	m26	2	2	2	2	2	2	1	2	2	2	1	2	2	2	2	2	214	113	2	554	210	135	2	580	232	132	2	557	234	117	1	551
1011011	m27	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	227	110	2	554	212	117	2	552	232	130	2	574	214	129	2	552
1011100	m28	2	2	2	2	2	2	3	2	2	3	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	3	2	2	2	2	2	2
1011101	m29	2	1	2	2	2	1	2	2	2	1	2	2	2	1	2	2	2	0	2	2	2	0	2	2	2	0	3	2	2	2	2	2
1011110	m30	2	1	2	2	2	0	2	2	2	0	2	2	2	0	2	2	2	1	2	2	3	1	2	2	22	1	2	2	2	1	2	2
1011111	m31	2	2	2	2	2	2	2	2	2	2	2	2	2	3	2	2	2	2	2	2	2	2	5	2	2	2	2	2	2	2	2	3
1100000	m32a	227	125	1	570	226	107	2	563	233	106	2	581	246	115	1	565	240	125	2	567	224	122	1	542	244	130	1	560	233	127	1	558
1100000	m32b	11	10	1	38	14	13	1	58	16	12	2	51	15	30	2	57	51	26	1	193	69	29	1	199	51	42	1	208	60	30	1	209
1100001	m33a	247	121	2	565	242	117	2	561	247	113	2	578	232	109	2	580	239	121	26	549	244	122	2	567	233	124	2	560	259	124	2	563
1100001	m33b	24	10	1	62	25	12	2	63	24	10	1	62	26	11	2	66	23	9	1	77	25	11	1	77	39	10	1	77	26	11	1	77
1100010	m34a	232	108	2	582	226	107	1	567	232	107	2	564	247	116	2	526	225	119	1	569	225	121	2	561	229	137	2	547	260	136	1	562
1100010	m34b	11	1	2	30	14	3	1	33	10	1	2	27	13	3	1	30	11	0	1	28	14	3	1	32	10	1	2	26	13	4	1	30
1100011	m35a	243	99	1	547	240	99	2	530	264	114	1	558	245	112	2	538	240	126	1	520	242	112	2	533	228	119	2	534	244	111	2	518
1100011	m35b	10	10	2	38	14	14	2	41	17	10	1	40	29	13	1	45	50	27	1	155	54	29	1	157	52	27	2	153	55	30	1	161
1100100	m36a	220	118	1	543	217	117	2	560	227	127	2	534	222	116	1	508	215	131	1	536	213	114	1	535	232	115	2	504	239	126	6	525
1100100	m36b	219	105	2	578	218	108	1	576	240	102	1	569	222	103	1	587	212	97	2	587	251	105	1	554	255	98	1	558	230	99	2	551
1100100	m36c	10	10	2	36	14	27	1	42	11	9	1	37	13	15	1	40	9	9	1	37	13	12	1	41	10	9	1	56	13	12	3	58
1100100	m36d	10	8	2	36	13	10	2	58	10	8	1	33	12	11	1	36	9	7	2	70	13	10	1	57	10	7	2	38	12	11	2	58
1100101	m37a	234	118	2	509	230	117	2	521	240	102	2	506	228	108	2	507	212	97	1	544	234	98	2	553	233	99	1	574	236	116	2	551
1100101	m37b	23	10	1	88	37	12	2	58	24	22	1	81	26	11	1	82	22	9	1	76	24	12	1	79	23	10	2	60	27	11	1	79
1100110	m38a	219	103	2	504	216	104	2	503	231	104	1	502	238	104	2	503	235	119	1	558	212	97	1	559	218	102	2	541	216	100	2	572
1100110	m38b	10	1	1	26	13	3	2	29	10	1	2	24	13	3	2	29	22	0	2	28	14	15	2	31	10	0	1	60	13	4	2	31

continued on next page

Table A.1: *CitiesMan* absolute (ms): average over all iterations of a query with DB intervals
(continued from previous page)

class	ID	+DB intervals																															
		+indirect keyword occurrences																−indirect keyword occurrences															
		+index intervals								−index intervals								+index intervals								−index intervals							
		+label index				−label index				+label index				−label index				+label index				−label index				+label index				−label index			
		ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG
		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
1100111	m39a	235	96	2	508	238	95	2	522	240	111	1	506	238	109	1	504	215	93	2	558	229	92	2	534	234	94	1	536	232	91	1	555
1100111	m39b	10	9	2	49	29	12	2	37	9	9	1	58	24	17	1	76	9	8	1	57	12	11	1	39	10	8	1	35	13	24	2	40
1101000	m40	2	2	2	2	7	8	1	7	1	2	1	2	7	7	2	6	19	6	1	31	25	11	1	38	20	7	1	33	24	12	1	37
1101001	m41	2	2	1	2	8	8	2	8	2	2	2	2	7	7	2	8	47	11	2	72	53	33	2	77	48	24	1	72	54	17	2	77
1101010	m42	1	2	1	3	8	7	2	9	1	2	1	2	12	8	2	7	40	20	1	75	29	26	1	92	23	20	1	78	28	25	2	80
1101011	m43	2	2	2	2	8	8	1	8	1	2	1	2	8	6	1	7	18	18	1	113	23	23	1	119	17	18	1	159	23	23	1	119
1101100	m44a	23	31	2	37	29	36	1	42	23	36	1	36	28	47	1	41	23	29	1	51	29	51	1	53	22	30	1	35	28	34	1	40
1101100	m44b	2	2	2	2	8	8	1	8	1	2	2	2	7	7	2	8	1	2	2	2	8	8	1	8	2	2	1	2	7	7	2	7
1101101	m45	2	0	1	2	8	7	2	7	2	0	2	1	7	6	2	6	1	1	2	2	8	6	1	8	2	1	2	2	7	6	2	7
1101110	m46	2	0	2	2	8	6	2	9	2	0	2	2	7	6	3	7	2	0	1	2	7	8	1	8	1	1	1	2	8	6	2	7
1101111	m47	1	2	2	2	8	8	1	8	2	2	1	2	7	7	2	8	2	2	1	2	7	8	2	9	1	2	1	2	7	7	2	8
1110000	m48	1	2	2	2	2	2	2	2	2	2	1	2	2	2	1	2	209	124	2	500	239	116	2	511	211	120	1	498	209	121	1	493
1110001	m49	2	2	1	2	2	2	1	2	2	2	1	2	1	2	1	2	226	116	1	492	224	115	1	494	228	129	2	522	209	116	2	492
1110010	m50	2	5	2	2	2	2	1	2	2	3	1	2	2	2	2	2	210	113	2	502	207	114	1	492	236	137	2	511	227	114	1	505
1110011	m51	2	2	1	2	2	2	2	2	1	2	1	2	1	2	2	2	225	107	2	528	210	110	1	514	231	109	2	514	227	121	1	489
1110100	m52a	15212	20309	1	22893	15079	20287	3	22677	15010	19374	1	22474	14875	19453	2	22478	14955	19195	2	22333	14676	19161	3	22177	14613	18904	2	21991	14442	18978	2	21857
1110100	m52b	58	56	2	192	54	65	1	208	57	50	2	207	57	50	1	174	54	69	21	206	54	48	2	211	56	48	1	189	54	48	2	197
1110101	m53	99	46	1	235	101	45	2	237	122	45	2	241	114	44	2	244	115	43	2	250	130	58	1	254	134	55	2	250	103	44	1	255
1110110	m54	55	52	2	192	55	52	1	207	56	52	1	188	57	52	2	204	54	64	2	191	54	49	2	190	56	49	1	191	55	50	2	189
1110111	m55	55	56	2	207	54	54	1	206	69	55	1	200	57	57	1	203	54	53	2	190	53	68	1	189	55	54	2	206	55	53	2	189
1111000	m56	1	2	2	2	1	2	2	2	2	2	1	2	2	2	1	2	226	135	1	572	225	118	2	554	231	121	1	551	233	133	1	532
1111001	m57	1	2	1	2	2	2	2	2	2	2	1	2	2	2	2	2	228	132	2	559	225	118	2	552	216	120	2	550	228	131	1	566
1111010	m58	1	2	2	2	2	14	2	2	2	3	2	2	2	2	2	2	219	136	1	551	217	115	1	573	232	116	1	557	215	129	1	555
1111011	m59	2	2	2	2	1	2	2	2	1	2	1	2	2	3	1	2	214	113	2	575	212	123	2	557	233	126	2	557	235	109	2	540
1111100	m60a	70	89	2	103	69	90	2	101	64	84	1	100	63	85	1	114	64	84	2	102	62	83	1	114	62	82	2	98	76	88	2	98
1111100	m60b	1	2	2	2	2	2	1	2	5	2	1	2	2	3	2	2	2	2	1	2	1	2	2	2	2	2	1	2	2	2	2	2
1111101	m61	2	0	2	7	2	0	2	2	2	1	1	2	2	1	2	2	1	1	2	2	2	0	1	2	2	0	1	2	2	1	1	2

continued on next page

Table A.1: *CitiesMan* absolute (ms): average over all iterations of a query with DB intervals
(continued from previous page)

class	ID	+DB intervals																															
		+indirect keyword occurrences																−indirect keyword occurrences															
		+index intervals								−index intervals								+index intervals								−index intervals							
		+label index				−label index				+label index				−label index				+label index				−label index				+label index				−label index			
		ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG
		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
1111110	m62	1	0	2	2	1	1	2	2	2	2	2	2	2	0	2	2	2	1	2	2	1	1	2	2	1	2	2	2	2	1	2	2
1111111	m63	2	2	2	2	1	2	2	2	1	2	1	2	2	2	1	2	1	2	1	2	2	2	1	2	2	2	2	2	1	2	1	2

Table A.2: *CitiesMan* absolute (ms): average over all iterations of a query without DB intervals

class	ID	−DB intervals																															
		+indirect keyword occurrences																−indirect keyword occurrences															
		+index intervals								−index intervals								+index intervals								−index intervals							
		+label index				−label index				+label index				−label index				+label index				−label index				+label index				−label index			
		ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG
		32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
0000000	0a	15524	20217	17526	25462	15450	19975	17350	25315	15033	19939	17159	25067	14943	19783	16994	24834	21242	28716	24313	34638	21054	28501	24029	34338	20910	28349	23776	34033	20355	28148	23458	33757
0000000	0b	7563	9824	8530	12340	7483	9806	8526	12265	7241	9720	8343	12162	7292	9702	8353	11990	10410	14089	11885	17039	10371	13945	11798	16898	10248	13876	11638	16770	10080	13818	11546	16639
0000001	1	2755	2838	2500	4369	2738	2803	2456	4362	2718	2800	2442	4344	2694	2807	2420	4290	3144	3430	2928	5007	3103	3403	2915	4976	3085	3410	2908	4906	3048	3368	2888	4877
0000010	2	5885	7462	6412	9699	5834	7389	6418	9613	5697	7369	6334	9563	5670	7334	6233	9423	8045	10759	9111	13297	7991	10671	8926	13168	7873	10522	8891	13064	7702	10557	8821	12911
0000011	3a	1383	884	788	2162	1344	881	777	2135	1352	905	792	2157	1358	885	783	2120	1388	919	805	2155	1379	914	800	2164	1350	942	822	2165	1333	953	780	2154
0000011	3b	604	443	383	987	626	445	407	1002	603	449	403	992	611	448	387	977	740	516	449	1204	757	495	435	1231	759	488	450	1193	724	508	440	1213
0000100	4a	8486	10657	9296	13638	8455	10562	9201	13502	8200	10502	9083	13403	8165	10461	9007	13262	8016	10325	8757	13187	7937	10265	8623	13071	7866	10189	8575	12977	7721	10121	8466	12826
0000100	4b	3403	4598	4001	5463	3405	4574	3948	5491	3308	4534	3903	5398	3323	4517	3873	5372	3199	4478	3779	5309	3249	4476	3750	5292	3166	4391	3712	5189	3124	4383	3657	5206
0000101	5a	1869	1634	1454	2919	1825	1634	1418	2885	1846	1630	1438	2873	1821	1619	1416	2843	1752	1571	1344	2845	1716	1591	1362	2809	1739	1555	1371	2777	1723	1579	1341	2771
0000101	5b	450	586	499	698	482	590	526	730	439	563	529	726	475	569	495	739	432	571	496	696	460	554	485	714	420	568	470	690	431	555	476	704
0000110	6a	3393	4146	3576	5540	3396	4114	3556	5508	3336	4095	3514	5474	3297	4079	3481	5427	3206	4007	3394	5378	3194	3948	3323	5347	3211	3956	3338	5296	3096	3923	3271	5216
0000110	6b	2773	3771	3274	4482	2747	3731	3204	4449	2663	3731	3201	4386	2631	3689	3140	4358	2650	3647	3080	4314	2623	3594	3045	4298	2544	3615	3025	4245	2538	3551	2999	4212
0000111	7a	1206	683	626	1894	1200	729	609	1894	1191	722	655	1896	1189	685	610	1873	1139	676	588	1854	1126	649	567	1860	1167	683	600	1832	1111	662	569	1844
0000111	7b	146	134	138	255	181	152	132	294	144	149	120	244	163	144	135	287	139	131	136	251	179	143	141	272	138	132	129	249	171	141	138	287
0001000	8	516	694	597	853	564	708	663	885	499	688	606	840	544	715	597	864	800	1035	879	1332	808	1050	893	1361	792	1026	864	1297	789	1006	876	1303
0001001	9	117	169	154	202	151	178	157	241	129	153	134	181	160	179	174	219	223	284	231	389	271	313	267	418	228	285	248	388	248	297	256	417
0001010	10	424	546	478	669	423	567	520	666	391	564	467	677	414	546	491	685	634	853	697	1046	664	834	729	1043	635	857	709	1029	640	822	733	1022
0001011	11	19	24	24	36	58	35	32	64	19	24	23	35	39	51	33	56	139	104	81	236	147	118	91	258	117	105	88	238	166	116	90	253
0001100	12	282	310	294	398	268	347	322	446	231	319	282	407	291	351	303	418	229	334	273	382	253	334	293	421	248	330	268	383	248	330	302	396
0001101	13	48	62	58	78	86	81	76	115	77	62	56	77	78	114	74	111	45	61	54	81	113	80	73	112	45	60	52	84	77	79	74	108
0001110	14	661	886	769	1023	681	858	773	1050	621	846	755	1019	639	848	753	1033	614	851	714	987	650	853	732	1015	638	840	718	985	612	848	702	1004
0001111	15	38	65	46	89	61	58	75	122	44	64	44	65	57	57	55	95	42	63	42	101	75	59	52	90	36	50	42	86	57	73	51	103
0010000	16	9	11	12	31	9	11	11	30	10	12	11	29	10	11	12	43	5900	7484	6333	9922	5830	7404	6244	9842	5819	7377	6201	9723	5657	7345	6099	9666
0010001	17	9	11	12	14	9	11	11	14	9	11	12	12	9	12	24	14	1438	1199	1017	2360	1405	1174	1006	2328	1427	1218	1015	2332	1396	1178	1036	2292
0010010	18	7	7	8	31	7	8	9	32	6	8	9	14	7	10	9	16	2506	3030	2554	4256	2478	2996	2527	4227	2448	2985	2510	4153	2399	2982	2479	4137
0010011	19	7	8	10	8	8	8	9	9	7	8	10	8	8	8	9	10	1047	633	538	1784	1052	590	516	1757	1108	601	527	1764	1048	618	518	1757
0010100	20	1769	2006	1806	2657	1752	1963	1807	2620	1736	1973	1711	2584	1698	1941	1765	2590	1659	1919	1711	2589	1622	1972	1696	2569	1654	1959	1721	2542	1634	1872	1657	2528

continued on next page

Table A.2: *CitiesMan* absolute (ms): average over all iterations of a query without DB intervals
(continued from previous page)

class	ID	−DB intervals																															
		+indirect keyword occurrences																−indirect keyword occurrences															
		+index intervals								−index intervals								+index intervals								−index intervals							
		+label index				−label index				+label index				−label index				+label index				−label index				+label index				−label index			
		ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG
		32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
0010101	21	411	384	347	623	385	379	361	635	386	382	341	620	380	379	341	600	371	375	345	608	371	372	326	606	371	373	329	599	389	368	327	606
0010110	22	858	962	845	1307	853	941	842	1304	844	923	776	1270	821	918	814	1281	810	901	790	1273	781	898	786	1265	817	899	778	1218	807	892	752	1259
0010111	23	276	188	159	441	286	171	153	432	278	159	139	430	272	170	158	442	255	150	132	434	274	150	148	429	281	168	134	457	262	150	168	442
0011000	24	9	11	12	46	10	12	12	29	10	12	12	29	15	11	17	29	5879	7456	6328	9913	5805	7438	6230	9835	5789	7375	6175	9735	5663	7312	6092	9666
0011001	25	10	11	12	13	9	11	12	13	9	11	11	14	9	11	12	12	1437	1183	1047	2358	1423	1174	1019	2351	1446	1205	1014	2338	1397	1198	1016	2328
0011010	26	6	7	8	15	21	7	9	15	6	7	8	15	8	8	8	13	2497	3045	2572	4239	2456	3015	2510	4210	2454	2984	2509	4177	2389	2983	2461	4145
0011011	27	7	9	9	9	7	8	9	9	7	8	9	9	7	10	9	8	1042	590	523	1782	1031	606	555	1757	1093	621	530	1750	1068	617	523	1747
0011100	28	177	246	193	280	171	210	217	281	166	208	173	246	147	218	193	264	145	212	184	256	145	207	171	260	181	192	180	251	164	205	164	267
0011101	29	41	53	49	88	41	53	49	65	41	53	47	66	40	52	49	68	40	52	47	65	43	52	46	65	40	51	45	83	57	51	45	63
0011110	30	421	513	446	606	395	493	439	576	374	492	460	588	365	491	432	579	361	481	427	576	349	481	417	553	361	498	412	603	376	481	409	581
0011111	31	34	43	40	63	39	43	40	59	34	43	39	61	32	42	40	58	48	41	39	62	32	42	63	61	32	41	38	79	33	44	40	56
0100000	32a	257	186	150	599	271	159	177	586	262	191	148	560	270	186	184	591	2024	2622	2249	3298	1994	2615	2197	3272	1995	2618	2198	3245	1939	2607	2176	3225
0100000	32b	13	13	12	28	16	15	16	27	13	14	14	39	16	15	15	27	103	99	90	187	106	103	95	192	117	106	87	188	118	118	91	170
0100001	33a	261	153	139	372	245	135	122	375	243	137	139	358	231	135	125	374	270	203	186	427	284	199	174	425	275	202	169	440	256	209	183	441
0100001	33b	24	17	16	55	25	24	19	40	24	16	16	39	26	18	18	40	24	17	16	38	25	17	19	38	24	16	15	38	30	18	17	38
0100010	34a	231	157	124	488	245	137	125	471	243	140	128	492	247	138	140	494	1149	1496	1262	1973	1138	1471	1245	1956	1161	1477	1210	1943	1125	1449	1199	1897
0100010	34b	14	24	12	27	17	15	15	27	16	13	12	25	18	15	15	29	14	12	15	23	18	18	18	59	14	11	12	23	20	15	15	27
0100011	35a	235	145	127	363	214	133	106	398	235	121	124	373	217	135	108	360	235	125	107	357	214	120	107	347	234	123	108	341	270	122	121	355
0100011	35b	10	8	9	20	13	11	12	22	11	10	9	19	13	11	12	22	52	23	22	93	64	30	25	93	55	27	22	89	53	27	30	91
0100100	36a	613	614	622	687	619	643	635	625	596	642	642	677	573	669	631	665	517	613	634	713	624	651	697	663	655	656	547	659	559	672	636	644
0100100	36b	236	150	144	419	240	159	148	407	251	146	138	429	248	155	131	409	233	146	144	405	250	142	142	423	235	140	129	404	235	133	122	416
0100100	36c	33	30	34	73	36	33	40	58	25	36	30	32	31	36	35	38	28	36	31	27	34	47	35	52	29	34	30	36	33	32	34	39
0100100	36d	71	87	80	103	73	89	82	107	68	105	101	102	71	91	82	105	67	99	92	100	86	87	88	108	67	84	76	115	90	102	77	101
0100101	37a	215	130	134	360	215	129	137	351	240	132	119	353	230	130	118	349	218	124	109	357	214	138	109	346	207	141	111	333	203	125	125	346
0100101	37b	39	17	19	37	39	18	17	52	24	16	16	37	25	17	19	39	22	16	15	40	23	18	16	38	22	16	15	36	24	17	17	38
0100110	38a	237	129	124	391	231	127	116	371	221	131	118	391	216	130	134	382	225	128	112	381	202	123	128	378	227	125	127	382	227	125	112	365
0100110	38b	14	11	12	20	18	37	16	24	14	11	11	22	17	15	15	24	13	11	12	24	18	19	16	24	13	12	12	22	18	16	18	25
0100111	39a	209	115	119	359	211	129	103	350	249	123	105	376	226	117	105	352	202	126	113	330	196	125	114	341	205	121	122	346	200	129	102	333

continued on next page

Table A.2: *CitiesMan* absolute (ms): average over all iterations of a query without DB intervals
(continued from previous page)

class	ID	−DB intervals																															
		+indirect keyword occurrences																−indirect keyword occurrences															
		+index intervals								−index intervals								+index intervals								−index intervals							
		+label index				−label index				+label index				−label index				+label index				−label index				+label index				−label index			
		ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG
		32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
0100111	39b	10	7	7	22	13	11	11	22	11	7	7	19	13	10	11	35	9	8	7	19	13	10	11	23	9	8	7	18	13	10	11	22
0101000	40	1	3	3	28	8	8	8	10	1	2	3	4	7	8	9	11	47	28	25	55	37	34	33	60	32	29	26	54	38	34	31	59
0101001	41	1	2	3	2	7	8	10	8	2	2	3	2	8	7	9	8	50	17	16	74	66	24	24	84	49	17	16	73	60	23	22	94
0101010	42	2	2	3	3	9	8	10	9	1	2	2	3	7	7	24	8	236	341	291	419	258	330	300	422	231	338	290	412	231	340	277	421
0101011	43	2	2	3	2	7	8	8	8	1	2	3	2	7	7	9	7	17	21	14	44	23	21	21	53	17	16	14	47	23	20	20	48
0101100	44a	11	15	20	17	18	22	20	23	12	15	13	17	17	21	19	22	11	15	14	22	17	27	19	22	11	16	13	15	17	20	17	21
0101100	44b	8	11	10	14	13	16	17	18	7	10	11	13	12	16	16	17	8	9	9	13	13	16	16	18	7	10	10	27	13	30	15	18
0101101	45	4	4	5	5	8	10	11	10	3	4	4	4	9	21	10	10	3	4	4	4	9	10	12	10	3	4	4	4	8	9	10	12
0101110	46	140	179	177	207	142	202	186	220	131	192	177	206	149	181	167	226	127	189	147	217	172	180	192	230	129	171	162	199	126	173	173	203
0101111	47	2	3	3	3	8	9	11	9	3	3	4	3	8	9	10	14	2	3	4	3	8	10	10	9	3	8	3	3	20	9	9	9
0110000	48	6	8	8	14	6	9	8	12	6	9	8	12	6	8	8	13	1475	1934	1662	2514	1487	1929	1625	2478	1484	1923	1623	2470	1447	1883	1586	2455
0110001	49	5	6	7	8	5	6	7	8	5	7	7	7	5	7	7	8	230	173	166	423	251	173	186	438	253	185	167	396	236	183	149	394
0110010	50	2	3	4	4	2	3	5	5	15	3	4	5	3	3	3	5	885	1109	944	1523	896	1081	914	1515	875	1100	906	1503	854	1085	895	1488
0110011	51	2	3	4	3	2	3	4	3	2	3	4	4	3	3	4	3	214	129	120	345	195	112	114	342	202	116	118	342	199	116	101	338
0110100	52a	8199	10537	9370	11824	8103	10399	9255	11735	7888	10400	9201	11723	7969	10299	9125	11533	7902	10216	8872	11555	7818	10225	8843	11498	7716	10043	8779	11369	7416	10022	8621	11246
0110100	52b	836	1047	928	1237	805	1008	922	1233	812	1039	916	1195	814	1009	923	1225	797	1026	896	1209	807	1052	882	1213	767	1012	873	1203	799	1001	860	1149
0110101	53	138	77	85	183	116	76	69	198	118	77	69	183	117	76	68	181	112	74	67	179	111	88	80	184	113	95	70	215	111	73	65	177
0110110	54	524	655	571	791	504	654	567	805	510	637	563	814	505	641	592	767	509	639	542	773	499	644	541	787	482	616	536	795	500	636	539	753
0110111	55	71	52	60	134	55	74	55	119	64	54	48	100	56	73	48	136	54	66	60	134	53	51	45	97	54	58	46	97	60	52	46	116
0111000	56	7	8	8	14	7	8	10	14	7	8	9	14	6	8	9	14	1493	1923	1613	2519	1474	1932	1667	2522	1454	1938	1608	2496	1441	1894	1589	2467
0111001	57	5	6	7	7	5	6	8	7	5	6	8	7	5	6	7	7	234	169	166	404	231	184	155	401	233	187	173	416	231	171	156	397
0111010	58	3	3	4	5	3	3	4	5	3	3	3	4	3	3	4	5	909	1114	918	1535	879	1134	916	1501	889	1096	909	1499	861	1091	900	1520
0111011	59	2	3	5	3	3	3	4	3	3	3	3	3	3	3	4	4	215	130	97	364	232	115	112	346	219	119	124	358	340	116	115	371
0111100	60a	34	43	39	49	33	43	38	49	30	43	38	49	47	43	38	48	33	42	37	51	33	42	37	47	31	42	36	47	59	40	34	46
0111100	60b	25	34	29	40	24	34	30	39	24	48	28	40	23	33	31	39	23	33	28	40	24	32	28	39	23	32	27	38	25	31	29	36
0111101	61	9	12	11	13	9	11	11	13	8	11	11	13	9	12	11	13	8	12	10	13	8	11	11	13	8	12	10	12	10	12	11	12
0111110	62	227	292	260	349	212	290	240	342	218	280	260	342	205	293	243	332	223	290	245	335	216	284	250	336	213	294	252	315	347	285	227	330

continued on next page

Table A.2: *CitiesMan* absolute (ms): average over all iterations of a query without DB intervals
(continued from previous page)

class	ID	−DB intervals																															
		+indirect keyword occurrences																−indirect keyword occurrences															
		+index intervals								−index intervals								+index intervals								−index intervals							
		+label index				−label index				+label index				−label index				+label index				−label index				+label index				−label index			
		ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG
		32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
0111111	63	5	7	7	6	6	7	7	6	4	18	7	7	4	6	6	7	4	7	7	6	4	6	7	6	4	5	7	7	17	5	6	6
1000000	m0a	233	149	2	353	231	125	4	343	242	151	14	348	218	124	3	346	215	141	3	355	226	125	2	354	217	144	2	333	227	138	2	347
1000000	m0b	248	151	2	353	217	138	2	357	239	128	2	348	255	124	2	362	250	123	2	357	232	137	2	356	248	136	2	347	218	123	2	331
1000001	m1	250	154	2	369	216	120	2	350	236	127	2	348	239	122	2	364	215	153	2	352	212	120	2	369	230	124	2	365	229	141	2	363
1000010	m2	220	123	2	353	243	117	2	356	226	133	2	366	219	135	2	369	234	117	2	374	227	117	2	350	220	140	3	347	244	118	2	335
1000011	m3a	233	148	2	371	233	131	2	368	227	135	3	365	235	149	2	345	217	115	2	369	222	114	2	346	228	117	2	365	242	116	2	350
1000011	m3b	219	131	2	351	216	114	2	369	241	117	2	348	220	137	3	359	232	133	3	368	216	114	3	348	231	117	2	343	244	116	2	360
1000100	m4a	243	125	2	356	241	120	2	347	265	138	2	346	263	121	2	349	235	133	2	356	199	132	3	338	203	134	2	339	218	139	2	337
1000100	m4b	23	34	2	53	25	18	2	45	22	16	2	37	25	18	2	38	36	17	2	37	23	16	3	40	22	16	2	37	24	33	2	38
1000101	m5a	216	135	2	348	214	118	2	360	219	136	2	364	212	118	2	379	220	114	2	346	212	114	2	346	211	132	2	355	220	115	2	341
1000101	m5b	22	13	2	52	40	14	3	58	22	14	2	37	24	14	1	38	22	13	2	37	23	14	2	40	37	12	2	36	23	14	2	55
1000110	m6a	227	116	1	352	213	116	2	349	248	133	2	352	230	115	2	343	205	128	1	358	234	120	2	340	210	114	2	338	236	131	2	368
1000110	m6b	11	0	1	17	31	3	2	21	10	0	1	19	14	2	1	36	11	1	2	16	13	2	2	21	10	1	1	15	14	2	2	20
1000111	m7a	228	128	2	367	226	113	2	347	233	120	2	349	218	131	1	348	206	126	2	341	202	130	2	340	205	124	2	340	223	145	2	340
1000111	m7b	24	12	2	53	24	13	2	38	22	12	2	66	24	14	2	38	23	11	2	36	23	12	2	50	21	12	2	54	24	12	2	37
1001000	m8	2	3	2	2	9	5	2	9	2	3	2	2	11	6	2	8	17	15	2	44	22	18	2	67	17	14	2	62	23	18	2	52
1001001	m9	2	2	2	2	10	5	2	9	2	2	2	2	9	5	2	8	20	3	2	27	25	6	1	37	34	4	2	27	25	6	2	34
1001010	m10	2	2	2	2	8	5	1	22	2	2	2	2	8	6	2	8	20	19	2	58	26	27	1	64	20	19	2	57	27	22	1	46
1001011	m11	2	3	2	2	9	4	2	23	2	3	2	2	8	5	2	9	20	17	2	62	22	18	2	50	16	18	2	44	23	18	2	84
1001100	m12	2	3	2	2	9	6	2	10	2	2	2	3	9	5	2	46	2	2	2	2	10	5	2	28	2	2	2	2	8	7	3	42
1001101	m13	2	1	2	2	9	4	2	9	2	1	2	2	7	3	3	8	2	0	2	2	9	4	2	9	1	0	2	22	23	3	2	9
1001110	m14	2	0	2	2	9	4	2	8	2	0	1	2	10	4	2	8	2	1	2	13	9	3	2	9	2	1	2	2	9	4	1	9
1001111	m15	2	3	2	2	9	5	2	15	2	3	2	2	9	5	2	9	2	2	2	2	10	5	2	12	2	3	1	2	8	5	2	9
1010000	m16	2	2	2	2	2	2	2	2	1	2	2	2	3	2	2	2	212	115	2	321	200	114	2	338	214	115	2	338	196	121	2	332
1010001	m17	2	2	2	2	2	2	2	2	2	2	2	2	2	2	1	2	194	113	2	353	208	116	2	355	233	114	2	338	212	119	2	314
1010010	m18	1	2	2	2	3	2	2	2	2	2	1	2	2	3	2	2	229	111	2	340	211	109	2	330	229	113	2	323	209	108	2	348
1010011	m19	1	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	215	108	2	336	211	106	2	332	199	142	2	332	195	126	2	331

continued on next page

Table A.2: *CitiesMan* absolute (ms): average over all iterations of a query without DB intervals
(continued from previous page)

class	ID	−DB intervals																															
		+indirect keyword occurrences																−indirect keyword occurrences															
		+index intervals								−index intervals								+index intervals								−index intervals							
		+label index				−label index				+label index				−label index				+label index				−label index				+label index				−label index			
		ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG
		32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
1010100	m20	69	66	2	111	52	69	2	92	53	50	4	110	69	48	3	95	51	47	2	109	50	48	3	93	51	48	2	105	50	64	2	93
1010101	m21	96	42	2	165	131	42	2	151	114	44	2	149	116	43	2	180	93	41	2	147	89	56	2	147	93	59	3	161	91	41	2	160
1010110	m22	53	55	2	108	52	47	1	95	54	49	5	94	53	49	2	110	51	46	2	95	52	53	2	92	51	66	2	115	66	48	2	107
1010111	m23	54	48	3	128	57	45	3	94	54	47	2	92	52	51	3	109	51	45	2	111	50	79	2	111	56	46	2	93	51	45	2	93
1011000	m24	2	2	2	2	1	2	2	2	2	2	2	2	2	2	2	3	207	113	2	340	207	114	2	350	199	113	2	316	197	129	2	330
1011001	m25	2	2	2	3	2	2	2	2	2	2	2	2	2	2	2	2	196	111	2	335	213	144	2	333	232	115	2	349	228	128	2	331
1011010	m26	2	2	1	2	2	2	2	2	1	2	2	2	2	2	2	2	210	123	2	358	198	113	1	336	197	111	1	319	210	126	2	314
1011011	m27	1	2	2	2	2	3	2	2	1	2	2	2	2	2	2	2	214	122	2	352	210	125	2	348	213	109	2	353	195	108	2	321
1011100	m28	2	2	2	2	2	2	2	2	2	3	17	2	14	2	2	5	2	2	2	2	2	3	2	3	2	2	2	2	2	2	2	2
1011101	m29	3	1	2	2	2	1	2	2	2	1	2	2	2	0	2	2	2	0	21	2	2	1	2	2	2	1	3	2	2	0	2	3
1011110	m30	2	1	3	2	2	0	2	3	2	1	2	2	2	1	2	2	2	0	2	2	2	1	2	2	2	1	2	2	2	2	2	2
1011111	m31	3	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	5	3	2	3	2	2	2	2	2	2	2
1100000	m32a	236	116	2	356	215	98	2	346	236	102	1	344	235	100	2	346	216	124	2	350	208	120	1	349	230	121	2	345	245	120	2	350
1100000	m32b	10	10	1	18	13	12	2	22	11	9	1	18	13	12	1	42	46	37	1	87	51	28	1	89	48	26	1	91	50	28	1	109
1100001	m33a	216	99	1	349	231	113	1	346	241	101	2	330	215	99	1	344	231	117	1	353	207	117	1	352	220	119	1	343	229	118	1	344
1100001	m33b	29	10	1	37	25	12	1	39	24	10	2	36	24	11	2	38	22	10	1	37	24	11	2	38	22	10	2	37	24	11	1	53
1100010	m34a	230	102	1	339	216	115	3	346	232	107	2	345	230	116	1	343	209	114	2	355	228	116	2	352	227	116	2	360	227	116	1	344
1100010	m34b	10	0	2	22	13	4	1	20	10	0	1	17	12	4	2	20	10	0	1	16	14	3	1	20	10	0	1	28	14	3	1	19
1100011	m35a	230	91	1	347	235	91	2	343	235	110	1	345	235	94	1	343	212	123	2	355	225	108	2	331	226	110	1	328	227	123	1	364
1100011	m35b	10	10	2	19	13	12	1	21	10	10	2	18	14	12	2	21	47	26	1	87	50	47	2	88	48	27	1	90	52	29	2	96
1100100	m36a	208	128	1	352	207	111	1	344	233	114	1	343	226	114	1	354	206	125	1	349	196	116	1	337	218	112	2	335	214	111	1	333
1100100	m36b	222	113	1	346	221	109	2	328	217	114	1	327	209	98	2	340	202	94	1	346	196	94	1	343	201	96	2	334	198	110	1	322
1100100	m36c	10	9	1	18	13	12	1	23	10	9	1	18	12	14	1	35	9	11	1	17	12	13	1	22	9	9	1	18	12	12	1	21
1100100	m36d	10	7	1	17	12	11	1	21	9	8	1	18	12	11	2	20	8	7	2	18	12	10	1	20	9	8	1	17	13	10	2	19
1100101	m37a	222	96	1	345	222	111	1	349	229	98	2	343	221	98	1	368	214	94	1	338	201	93	2	335	222	95	2	332	214	114	1	332
1100101	m37b	22	15	2	36	24	12	1	37	22	10	1	36	24	10	1	38	20	9	1	36	23	11	2	37	21	10	2	36	37	10	2	36
1100110	m38a	208	99	2	345	206	112	2	335	213	134	2	343	217	114	2	339	201	110	1	323	195	95	2	340	222	97	1	333	198	111	2	318
1100110	m38b	10	0	2	17	13	3	2	19	10	0	1	16	12	4	1	19	9	0	1	16	13	4	2	19	9	0	2	16	13	3	2	18

continued on next page

Table A.2: *CitiesMan* absolute (ms): average over all iterations of a query without DB intervals
(continued from previous page)

class	ID	−DB intervals																															
		+indirect keyword occurrences																−indirect keyword occurrences															
		+index intervals								−index intervals								+index intervals								−index intervals							
		+label index				−label index				+label index				−label index				+label index				−label index				+label index				−label index			
		ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG
		32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
1100111	m39a	242	90	2	346	212	90	2	349	234	107	1	342	225	92	1	337	214	88	2	327	213	91	1	377	212	105	1	334	220	90	2	331
1100111	m39b	9	8	1	17	13	11	1	19	9	8	2	32	12	11	2	24	8	8	1	17	12	11	1	20	9	8	1	16	12	11	14	19
1101000	m40	2	2	1	2	7	8	1	7	1	2	2	2	6	7	1	8	17	6	2	28	38	12	1	32	18	6	1	26	24	12	1	31
1101001	m41	1	2	2	2	8	8	1	8	2	2	2	2	12	7	1	7	44	11	1	64	49	18	2	69	59	12	2	78	65	17	1	67
1101010	m42	2	2	2	2	7	8	1	8	2	2	1	1	6	7	1	7	20	19	1	55	29	41	2	45	22	19	1	39	27	25	2	44
1101011	m43	1	2	2	1	9	8	2	8	2	2	2	2	7	7	1	8	16	18	2	43	22	23	2	49	28	18	1	42	22	23	1	49
1101100	m44a	28	16	1	21	19	23	1	26	12	16	2	20	17	22	2	24	12	17	1	19	18	22	1	27	12	17	1	20	16	22	2	25
1101100	m44b	2	17	2	2	7	8	1	8	2	2	2	2	7	7	2	7	2	2	2	2	8	8	1	8	2	2	1	2	7	7	2	7
1101101	m45	1	0	2	2	7	6	1	23	1	0	1	2	7	9	1	7	2	1	2	2	8	7	1	8	1	0	1	2	7	5	1	7
1101110	m46	2	0	1	2	28	8	2	8	2	0	3	2	7	6	1	7	1	1	1	2	8	6	2	8	1	0	1	2	7	6	1	20
1101111	m47	2	2	1	2	7	8	2	8	2	2	1	2	7	7	2	7	2	2	1	2	9	29	2	8	2	2	1	2	7	8	2	6
1110000	m48	2	2	2	2	2	2	1	2	2	2	1	2	2	2	1	2	215	113	1	332	192	112	2	330	212	131	2	329	209	134	1	334
1110001	m49	2	2	1	2	2	2	1	2	2	2	2	2	1	2	2	2	208	110	1	338	194	127	1	330	196	113	1	357	208	112	1	328
1110010	m50	1	2	2	2	2	2	1	2	1	2	1	2	1	2	1	2	208	110	2	333	192	124	1	331	218	111	2	335	209	109	1	334
1110011	m51	1	2	2	2	2	2	1	2	2	2	1	2	2	2	1	1	217	102	2	324	195	101	1	338	213	120	1	336	210	113	1	329
1110100	m52a	8046	10548	2	11740	8057	10457	1	11763	7841	10365	1	11705	7830	10232	13	11501	7786	10233	1	11560	7835	10105	1	11490	7689	10063	1	11409	7427	10025	2	11270
1110100	m52b	53	48	1	99	53	47	1	94	55	48	2	109	53	47	1	93	49	46	1	94	64	46	2	109	66	47	2	109	70	53	2	107
1110101	m53	93	43	1	164	109	42	2	148	98	44	1	150	97	43	2	162	91	41	2	180	90	41	1	160	91	42	2	143	92	41	1	160
1110110	m54	52	54	1	114	52	50	1	114	54	57	2	94	54	49	2	124	50	47	1	92	66	47	1	108	51	49	2	91	67	49	1	117
1110111	m55	52	53	2	96	71	52	1	95	54	53	2	93	72	52	2	92	51	50	1	93	51	50	2	94	51	51	2	91	61	51	7	91
1111000	m56	1	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	212	114	1	328	209	114	1	333	227	117	2	332	229	131	2	330
1111001	m57	2	2	1	2	1	2	2	2	2	2	2	2	2	2	2	2	213	113	2	335	216	112	1	323	214	115	1	338	196	128	1	331
1111010	m58	1	2	1	2	2	2	2	2	2	2	1	2	2	2	1	1	222	140	2	336	214	110	1	324	204	115	2	336	227	126	1	331
1111011	m59	2	2	1	2	2	2	2	2	2	2	2	2	2	2	1	2	200	104	1	335	197	117	1	357	198	104	2	337	218	104	1	350
1111100	m60a	33	44	1	53	34	44	1	67	32	49	2	52	33	44	1	52	32	43	1	53	39	43	1	50	32	43	2	50	40	59	2	64
1111100	m60b	2	2	1	2	2	2	2	2	2	2	1	2	2	3	2	2	2	2	1	3	2	2	2	2	1	2	2	2	2	2	1	2
1111101	m61	2	0	2	2	1	0	1	2	1	0	2	2	2	0	2	2	2	1	2	2	2	0	1	2	2	0	2	2	6	1	2	2

continued on next page

Table A.2: *CitiesMan* absolute (ms): average over all iterations of a query without DB intervals
(continued from previous page)

class	ID	−DB intervals																															
		+indirect keyword occurrences																−indirect keyword occurrences															
		+index intervals								−index intervals								+index intervals								−index intervals							
		+label index				−label index				+label index				−label index				+label index				−label index				+label index				−label index			
		ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG
		32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
1111110	m62	2	1	1	3	1	1	1	2	2	0	1	2	2	1	3	2	2	0	1	3	2	1	1	2	2	1	1	2	2	1	1	2
1111111	m63	1	2	2	2	2	2	1	2	2	2	2	2	1	2	2	2	2	2	2	2	2	2	2	2	1	2	1	2	4	2	2	3

Table A.3: *CitiesAuto* absolute (ms): average over all iterations of a query with DB intervals

class	ID	+DB intervals																															
		+indirect keyword occurrences																−indirect keyword occurrences															
		+index intervals								−index intervals								+index intervals								−index intervals							
		+label index				−label index				+label index				−label index				+label index				−label index				+label index				−label index			
		ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG
		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0000000	101	302	195	163	769	313	210	237	814	340	204	200	820	332	224	235	799	4745	4618	4655	5450	5098	4931	4920	5352	5296	5091	5381	5793	5728	5415	5276	5984
0000000	102	884	472	538	2121	962	512	541	2115	1006	550	554	2110	1024	545	539	2146	5628	5293	5393	7005	6125	5482	5521	6878	6314	5826	6098	7487	6975	6000	6397	7800
0000001	104	451	203	209	961	486	213	242	1046	515	244	252	1049	499	239	260	1036	689	558	565	1041	738	556	582	1035	749	601	655	1148	773	637	643	1149
0000001	105	1005	472	549	2125	1054	522	529	2143	1148	557	551	2217	1171	560	566	2236	2098	1480	1444	3684	2205	1486	1492	3580	2362	1617	1602	3891	2548	1777	1680	3824
0000010	108	233	188	206	328	221	204	237	327	222	235	214	314	239	216	225	307	260	246	261	403	303	286	289	366	322	294	326	364	325	415	308	406
0000011	109	507	240	278	1034	527	231	235	1037	569	242	269	1076	588	251	279	1030	780	418	394	1632	826	357	385	1497	897	451	451	1633	887	476	427	1625
0000100	110	57	53	53	52	64	65	89	84	60	120	58	56	99	94	71	65	67	70	65	63	74	73	76	71	70	67	72	67	78	79	79	86
0000101	111	290	187	183	625	330	245	198	637	355	196	199	647	338	201	207	657	776	491	469	1354	805	539	512	1371	815	552	585	1398	856	554	530	1424
0000111	112	206	101	105	460	225	105	107	501	262	113	113	537	267	126	119	480	267	114	146	561	295	117	121	528	271	125	139	493	306	164	136	547
0001000	113	315	184	197	696	322	209	235	765	331	227	203	721	348	248	232	751	1728	1588	1631	2200	1805	1592	1594	2061	1919	1821	1880	2361	1903	1795	1740	2262
0001001	114	337	216	221	580	380	261	238	691	377	235	241	666	388	251	278	631	690	579	552	1027	719	581	583	994	763	608	624	1157	783	625	663	1060
0001001	115	257	153	159	518	258	146	149	561	285	140	166	559	306	181	158	586	701	526	567	992	719	580	587	971	750	631	629	1062	769	656	668	1028
0001011	118	269	124	128	440	291	140	141	498	298	140	142	492	313	148	154	488	289	153	171	567	253	113	115	456	338	200	167	559	316	148	134	555
0001100	119	164	190	171	188	188	193	196	190	181	178	204	200	213	226	202	230	197	226	217	239	244	250	254	221	209	223	221	210	229	236	245	281
0001100	121	13	17	16	11	21	24	22	18	16	15	15	12	22	22	28	18	16	16	16	12	26	24	27	21	17	17	20	15	24	27	24	23
0001100	122	10	12	11	10	17	18	18	16	11	11	11	11	18	16	17	17	12	13	12	13	18	50	18	19	12	14	12	13	18	21	19	18
0001101	123	299	158	172	492	316	207	237	506	331	175	174	524	381	212	216	539	333	191	182	626	349	219	197	569	382	227	200	553	350	239	242	564
0001101	124	25	47	22	38	40	33	36	46	31	23	24	32	44	35	35	47	31	26	26	36	44	40	38	52	33	29	31	66	46	39	43	53
0001110	125	234	211	222	253	230	260	238	246	259	256	263	322	250	250	256	262	288	315	294	345	270	277	300	305	271	310	295	313	285	302	323	333
0001110	127	457	456	462	481	466	527	525	510	538	510	550	531	568	572	556	549	549	583	610	639	569	594	741	622	637	650	706	666	740	626	728	714
0001111	128	548	555	554	558	557	583	591	610	637	620	618	648	635	632	645	706	656	665	679	697	702	726	723	723	743	741	815	757	768	816	782	774
0001111	129	5	5	8	30	12	14	52	35	5	6	7	34	14	11	13	38	6	6	7	63	13	13	14	41	7	7	7	51	15	17	14	38
0001111	130	533	540	561	567	559	594	590	608	601	640	608	626	672	631	654	654	637	656	690	695	670	703	834	720	816	745	746	833	752	891	758	805
0010000	131	33	31	31	99	59	32	34	99	37	33	33	155	39	33	36	111	3247	2990	3360	4137	3883	3186	3492	4099	3948	3384	3829	4424	3786	3429	3495	4524
0010000	132	93	128	94	189	93	122	128	207	155	94	123	199	168	104	96	209	3494	3256	3260	4428	3684	3390	3462	4373	3986	3593	3710	4995	3867	3906	3685	4622
0010000	133	160	148	174	205	147	167	193	163	210	213	139	233	171	163	153	203	1241	1125	1097	1670	1278	1274	1188	1628	1399	1192	1201	1735	1529	1297	1215	1700
0010001	134	611	418	534	876	567	567	568	906	696	559	518	936	707	476	523	996	1241	944	895	1767	1336	1094	1003	1778	1292	1079	1004	2004	1435	1002	984	1873
0010001	136	286	248	270	389	232	257	279	337	301	221	283	412	304	222	226	353	842	580	663	1289	837	578	623	1286	942	617	621	1337	985	626	606	1415

continued on next page

Table A.3: *CitiesAuto* absolute (ms): average over all iterations of a query with DB intervals
(continued from previous page)

class	ID	+DB intervals																															
		+indirect keyword occurrences																−indirect keyword occurrences															
		+index intervals								−index intervals								+index intervals								−index intervals							
		+label index				−label index				+label index				−label index				+label index				−label index				+label index				−label index			
		ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG
		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0010010	138	3067	2980	3037	3657	3269	3145	3187	3892	3445	3274	3318	3985	3561	3359	3456	4144	6649	6383	6535	7756	7734	6697	6991	7606	7670	7231	7119	8809	7792	7636	7257	8754
0010011	140	255	128	122	639	270	122	127	667	288	133	135	667	298	138	139	684	1508	721	733	3127	1600	740	765	3151	1881	845	811	3307	1910	810	806	3256
0010100	141	37	37	39	80	37	39	40	84	41	39	40	92	39	41	41	98	41	39	40	115	40	47	63	88	47	44	48	128	44	45	46	90
0010100	142	39	38	39	72	41	39	41	76	69	41	43	77	45	42	42	105	44	41	41	83	45	42	42	82	48	46	47	83	55	44	46	114
0010100	143	40	35	34	77	36	36	36	109	35	36	39	81	37	38	39	80	35	35	61	82	35	46	38	83	38	39	42	97	45	39	40	87
0010101	144	2424	2340	2418	2747	2593	2503	2521	2979	2773	2633	2638	3010	2861	2717	2760	3154	2977	2840	2875	3388	3207	2957	3002	3555	3348	3220	3174	3780	3356	3268	3368	3800
0010110	145	469	469	478	558	463	499	508	581	515	544	495	589	528	543	513	611	553	562	606	735	652	558	575	724	656	622	630	727	628	630	639	719
0010110	146	160	135	138	204	193	142	144	252	176	144	177	275	185	154	178	226	191	170	167	247	223	170	172	279	253	176	202	274	214	188	187	256
0010110	147	11	8	8	27	19	12	12	34	13	8	9	29	19	11	12	36	12	8	10	31	20	12	12	40	15	9	10	30	41	12	14	38
0010111	148	181	125	128	182	128	165	159	187	195	162	139	223	141	167	146	228	147	143	175	273	153	152	155	255	165	161	160	257	160	166	166	254
0010111	149	62	60	57	52	60	62	62	59	62	63	71	58	64	65	65	64	93	88	69	69	69	74	75	72	77	73	82	79	74	75	104	72
0010111	150	111	70	100	124	96	73	77	153	106	80	81	131	148	80	93	163	109	86	87	145	113	89	110	151	152	92	99	176	123	100	98	180
0011000	151	237	214	216	247	260	249	233	269	251	236	265	296	253	248	279	298	277	254	287	343	278	270	313	335	349	327	289	310	333	293	298	414
0011000	152	170	160	137	329	224	130	133	344	212	157	133	376	214	170	176	343	2394	2216	2224	2850	2449	2304	2450	2990	3056	2502	2510	3063	2698	2570	2626	3218
0011000	153	1383	1411	1486	1544	1448	1526	1503	1588	1583	1562	1574	1716	1627	1654	1611	1762	3846	3733	3743	4549	4179	4047	4122	4678	4586	4235	4196	4785	4708	4346	4493	5421
0011001	155	71	77	96	73	82	75	110	77	79	119	99	87	80	78	114	81	410	281	334	718	498	325	319	732	499	346	348	728	490	318	388	838
0011001	156	1996	2068	2068	2137	2157	2171	2147	2233	2199	2285	2186	2339	2287	2330	2346	2449	2452	2504	2564	2737	2755	2584	2629	2859	2701	2824	2712	2818	2882	2909	2770	3204
0011010	157	282	137	141	624	273	166	148	640	279	161	186	671	296	157	185	671	2501	2399	2444	3011	2737	2506	2528	3121	3101	2872	2739	3288	2992	2968	2797	3474
0011011	158	1663	1614	1634	1661	1699	1744	1726	1799	1798	1750	1793	1787	1828	1848	1850	1884	2151	2042	2079	2572	2234	2155	2357	2652	2386	2251	2266	2906	2665	2646	2288	2877
0011011	159	27	27	28	60	27	26	30	33	28	28	29	33	29	30	30	32	274	179	168	587	284	160	171	593	321	211	172	545	354	201	207	631
0011011	160	14	11	12	14	16	13	14	46	15	13	14	16	15	13	14	16	235	102	105	539	279	105	111	557	293	116	182	486	282	123	144	578
0011100	161	437	441	486	456	466	472	511	493	500	522	494	504	502	508	489	521	527	541	545	579	590	585	574	626	595	597	586	602	651	742	599	600
0011100	162	425	446	458	436	514	514	478	490	502	470	500	510	508	553	493	525	541	568	586	577	562	603	601	606	778	584	600	652	693	641	633	660
0011100	163	47	78	48	51	53	52	57	53	52	52	52	77	58	61	60	57	54	56	56	58	62	65	96	66	62	62	63	59	98	70	67	66
0011101	164	2009	2047	2106	2145	2158	2137	2177	2231	2225	2249	2253	2290	2313	2431	2344	2454	2400	2406	2464	2632	2718	2767	2499	2669	2925	2849	2707	2718	2691	2681	2927	2965
0011101	165	9	7	11	12	9	10	9	14	10	11	9	12	10	8	9	16	9	9	9	13	9	8	9	17	10	9	9	17	10	9	11	12
0011110	166	440	460	497	503	490	519	493	502	511	510	555	581	503	504	567	596	555	563	565	588	618	690	594	671	748	631	669	658	636	666	649	696
0011110	167	64	35	37	88	51	40	75	93	43	40	40	62	110	71	45	70	46	43	46	127	74	56	50	79	60	46	48	101	87	60	55	83

continued on next page

Table A.3: *CitiesAuto* absolute (ms): average over all iterations of a query with DB intervals
(continued from previous page)

class	ID	+DB intervals																															
		+indirect keyword occurrences																−indirect keyword occurrences															
		+index intervals								−index intervals								+index intervals								−index intervals							
		+label index				−label index				+label index				−label index				+label index				−label index				+label index				−label index			
		ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG
		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0011110	168	42	40	40	63	46	44	44	65	46	75	44	66	51	77	49	67	48	47	81	73	63	53	51	75	64	52	76	73	57	50	58	74
0011111	169	27	53	24	33	38	27	28	45	28	23	25	36	36	30	31	46	31	26	26	40	46	34	32	52	56	29	31	40	41	33	39	52
0011111	170	28	25	26	38	30	27	28	69	33	28	29	62	33	29	32	42	35	31	32	45	42	33	33	48	40	35	33	46	41	38	37	47
0011111	171	42	38	38	50	43	36	40	53	43	40	42	53	43	40	41	55	47	44	45	58	50	43	45	58	52	50	48	60	48	48	47	63
0100000	174	71	63	64	123	82	95	72	163	77	69	71	128	87	74	78	136	170	161	164	256	147	120	93	132	169	163	158	195	105	91	102	168
0100001	175	282	226	199	241	261	211	216	263	254	230	231	322	268	256	233	307	129	116	128	144	178	156	124	157	142	143	161	145	181	129	142	159
0100001	176	268	125	139	572	307	135	163	593	292	138	194	607	331	146	203	603	706	608	579	1055	835	612	613	1033	793	671	642	1141	874	637	812	1107
0100001	177	421	305	303	930	484	287	257	983	465	317	264	1019	511	319	276	1000	972	708	761	1604	994	695	731	1543	1170	795	796	1781	1229	790	765	1559
0100010	178	288	131	133	620	262	146	147	714	270	144	146	658	283	158	158	677	2506	2360	2726	3007	2637	2681	2699	3146	2983	2991	2646	3259	2882	2671	2884	3593
0100010	179	1006	925	946	1458	1091	932	1003	1524	1206	1037	1012	1566	1184	1065	1045	1573	3396	3317	3523	4044	3995	3684	3591	4104	3855	4121	3789	4222	4060	3758	4001	4621
0100010	180	261	226	231	368	285	247	304	364	256	278	282	362	273	293	304	388	308	329	320	388	292	404	328	375	367	370	302	458	398	336	318	464
0100011	181	38	30	57	45	46	36	39	48	46	31	66	43	48	34	48	49	29	30	33	32	64	35	33	45	31	35	33	33	45	35	63	41
0100011	182	11	10	11	11	18	16	15	17	11	10	12	11	18	16	16	19	10	11	14	11	17	20	51	27	11	14	14	11	19	18	20	17
0100011	183	229	101	104	492	229	117	148	487	293	114	116	509	308	127	129	547	281	126	133	513	289	125	132	552	280	149	165	564	303	136	137	527
0100100	184	37	32	34	96	75	37	41	104	40	36	38	70	53	43	48	81	44	38	46	123	55	52	48	88	46	44	44	77	58	44	48	143
0100100	185	96	99	98	98	109	138	144	111	107	108	133	107	116	117	119	118	113	114	125	123	154	167	158	136	165	156	139	157	148	163	170	137
0100100	186	49	18	18	34	61	25	28	42	23	18	22	65	32	26	28	44	27	22	22	40	34	29	29	78	29	23	26	40	34	26	28	51
0100101	187	65	43	45	221	73	51	50	182	71	48	50	186	79	52	56	158	77	51	82	193	89	69	67	199	86	56	62	227	93	62	87	203
0100101	188	20	15	16	30	26	24	24	36	21	17	19	61	37	24	22	65	23	43	18	34	31	27	26	45	24	18	20	37	29	26	27	42
0100101	189	212	111	139	503	250	144	119	491	241	157	133	509	301	128	130	519	279	124	133	555	282	133	166	563	277	135	145	552	313	147	142	554
0100110	190	30	17	17	71	35	23	25	52	28	18	19	45	37	25	26	53	33	19	22	53	38	29	28	56	33	21	48	51	42	26	29	64
0100110	191	584	600	591	637	637	667	612	719	638	625	637	708	655	653	678	740	710	712	731	845	719	914	736	879	814	922	775	874	840	807	809	1002
0100111	192	24	22	22	103	34	26	28	81	31	25	26	103	34	28	58	82	32	26	29	91	35	30	31	119	31	29	29	98	47	31	40	118
0100111	193	25	18	19	41	32	23	24	50	28	20	21	44	33	24	28	50	29	20	49	48	83	28	59	55	32	39	23	81	63	28	28	54
0100111	194	21	9	10	29	27	17	18	36	23	11	12	31	31	17	17	37	24	12	13	56	46	17	20	40	29	13	13	33	33	18	19	45
0101000	196	359	166	235	696	334	210	212	708	364	211	212	705	357	192	200	722	2322	2217	2421	2883	2785	2441	2480	3008	2802	2590	2571	2945	2769	2545	2552	3126
0101000	197	250	178	146	656	300	158	187	687	280	156	183	653	296	194	200	697	1655	1535	1672	2158	2004	1652	1657	2188	1938	1724	1752	2211	1992	1900	1780	2238
0101001	198	249	169	149	558	269	212	189	572	301	186	190	577	292	170	198	593	693	610	638	985	736	635	615	1052	798	642	683	1039	824	655	658	1191

continued on next page

Table A.3: *CitiesAuto* absolute (ms): average over all iterations of a query with DB intervals
(continued from previous page)

class	ID	+DB intervals																															
		+indirect keyword occurrences																−indirect keyword occurrences															
		+index intervals								−index intervals								+index intervals								−index intervals							
		+label index				−label index				+label index				−label index				+label index				−label index				+label index				−label index			
		ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG
		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0101001	199	237	129	130	525	257	132	135	557	261	172	146	572	294	146	150	577	682	554	580	954	705	586	563	1052	808	724	627	1020	788	629	622	1122
0101001	200	232	119	122	523	246	130	166	580	285	130	154	555	296	197	142	566	723	602	594	1014	722	596	613	1073	835	665	643	1036	792	688	649	1123
0101010	202	234	153	153	638	307	136	167	663	263	136	163	636	347	200	178	664	2469	2350	2556	3024	2892	2593	2466	3169	2944	2760	2765	3187	2873	2759	2671	3178
0101011	204	913	840	875	1212	982	859	938	1297	1057	936	916	1266	1105	951	1014	1410	1160	1012	1056	1514	273	110	112	542	1307	1222	1165	1717	354	118	122	553
0101011	205	23	16	18	37	25	18	20	38	27	18	19	44	28	20	20	40	34	25	60	54	26	17	21	66	37	29	30	79	29	18	20	40
0101011	206	247	101	105	500	238	116	119	510	272	115	117	505	252	155	129	563	284	126	165	580	285	134	151	516	301	146	149	571	339	135	136	551
0101100	207	3	4	4	1	10	29	8	7	2	5	3	1	10	8	8	6	4	2	2	2	8	9	9	7	2	2	3	1	11	10	8	7
0101100	208	6	7	7	5	14	14	15	13	7	7	7	6	14	14	14	13	7	15	9	6	15	15	16	20	9	8	8	8	15	16	15	14
0101100	209	14	16	14	15	20	20	19	19	46	14	47	14	19	19	21	19	17	14	20	17	21	21	20	21	20	21	15	15	21	20	21	21
0101101	210	3	0	1	5	9	15	9	10	3	0	2	4	9	6	8	10	4	0	2	7	9	6	8	12	4	0	2	5	9	7	8	11
0101101	211	1	1	3	2	9	8	9	9	2	1	3	2	7	7	9	8	2	1	3	2	8	8	10	9	1	3	3	3	7	8	9	8
0101101	212	230	99	103	487	271	104	107	480	233	111	139	505	303	119	116	545	238	112	130	544	296	117	117	558	299	151	194	567	269	145	153	553
0101101	213	2	2	2	3	8	8	9	9	2	2	3	3	7	7	9	8	1	2	2	3	8	8	11	9	2	2	4	2	8	9	9	36
0101110	214	20	13	15	51	28	18	21	55	24	15	16	48	28	19	73	55	22	17	16	51	32	19	21	89	28	46	17	55	30	54	22	54
0101110	215	19	41	17	64	24	48	42	62	50	15	17	65	23	48	21	69	22	17	18	38	26	49	21	38	22	18	21	38	26	20	22	71
0101110	216	24	54	24	28	32	31	39	37	27	28	26	33	35	33	37	39	27	28	30	35	57	35	39	45	31	31	31	35	37	39	38	47
0101111	217	58	53	65	83	96	59	61	126	70	59	60	90	102	65	65	98	70	65	69	129	107	71	73	110	76	69	72	133	108	74	79	110
0101111	218	9	5	7	13	12	10	11	19	9	6	8	11	12	10	12	16	9	7	8	13	12	11	12	17	9	7	8	15	14	11	12	18
0101111	219	3	4	4	3	9	10	11	11	3	3	33	3	9	9	10	9	3	3	5	5	11	10	11	10	4	3	5	3	9	10	11	10
0110000	220	340	214	217	721	355	277	225	758	348	257	257	773	357	269	266	780	1650	1259	1336	2516	1796	1536	1340	2623	1920	1467	1445	2803	2013	1449	1426	2915
0110000	221	793	783	805	1007	830	754	849	1025	858	854	854	1095	817	876	903	1084	3004	2688	2821	3748	3275	2821	2862	3855	3416	3044	3046	3963	3461	3348	3110	4017
0110000	222	619	619	626	790	684	613	657	833	690	680	681	846	671	701	696	918	2152	1918	1973	2729	2257	1942	1963	2895	2375	2169	2102	3140	2495	2163	2122	3161
0110001	223	183	125	129	420	158	131	135	423	190	138	140	433	177	142	153	404	201	158	166	551	233	168	191	516	251	196	174	564	238	178	180	594
0110001	225	191	163	163	209	166	136	195	209	170	167	168	212	142	169	169	249	547	406	418	896	593	393	390	862	593	501	443	921	599	426	491	900
0110010	226	319	210	178	692	301	161	163	678	296	224	171	694	303	204	176	698	2552	2407	2593	3041	2845	2593	2565	3177	3038	2771	2724	3188	2912	2964	2828	3272
0110010	227	657	690	659	809	673	630	695	848	727	754	723	870	748	743	765	914	1005	877	926	1309	998	987	930	1398	1182	969	967	1384	1232	1087	975	1465
0110010	228	624	622	670	808	666	688	700	857	736	683	735	840	717	720	754	933	950	842	896	1318	999	947	911	1386	1157	962	975	1366	1165	1038	984	1418
0110011	229	68	62	35	51	70	35	36	78	44	35	37	54	54	36	38	55	50	41	45	88	80	42	49	62	55	78	44	63	56	73	46	64

continued on next page

Table A.3: *CitiesAuto* absolute (ms): average over all iterations of a query with DB intervals
(continued from previous page)

class	ID	+DB intervals																															
		+indirect keyword occurrences																−indirect keyword occurrences															
		+index intervals								−index intervals								+index intervals								−index intervals							
		+label index				−label index				+label index				−label index				+label index				−label index				+label index				−label index			
		ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG
		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0110100	230	24	23	25	77	31	23	26	78	26	24	26	49	32	25	27	80	25	26	28	53	54	24	27	53	27	27	34	53	27	29	28	79
0110100	231	51	53	52	52	55	57	56	56	58	83	59	57	72	59	61	60	62	64	66	68	69	79	67	92	77	75	69	70	69	73	73	71
0110100	232	26	27	29	57	27	29	30	55	28	29	29	56	31	51	31	57	30	29	32	93	29	36	31	60	36	35	33	63	32	32	35	64
0110101	233	173	209	231	178	183	184	190	221	223	192	225	232	225	230	233	209	211	229	222	230	245	228	219	229	277	250	227	248	253	251	261	240
0110101	234	13	12	13	16	12	11	13	15	13	12	14	16	14	14	15	70	15	12	14	18	13	13	13	17	17	22	14	18	16	16	13	19
0110101	235	12	11	13	12	13	32	13	15	12	14	14	13	12	13	15	14	13	14	16	37	15	14	45	15	16	18	17	15	17	15	17	15
0110110	236	388	420	375	376	439	398	394	412	432	462	466	445	472	457	457	439	473	502	519	539	514	580	469	522	545	604	505	546	532	572	548	583
0110110	237	1134	1121	1107	1325	1208	1178	1165	1431	1315	1235	1267	1412	1331	1282	1299	1467	1419	1357	1437	1669	1493	1478	1391	1685	1575	1553	1479	1717	1663	1513	1573	1764
0110110	238	844	836	896	961	898	853	874	1093	929	934	971	1117	973	963	984	1129	1040	1003	1106	1200	1062	1047	1072	1282	1192	1205	1096	1293	1146	1150	1224	1293
0110111	239	14	20	12	13	14	12	13	17	13	13	13	16	15	14	14	17	15	14	15	16	14	12	15	17	15	13	17	18	15	14	17	18
0110111	240	18	15	16	32	18	16	15	28	21	16	17	28	21	18	18	43	21	18	20	30	22	17	19	32	23	18	21	30	28	19	50	31
0110111	241	133	74	134	220	141	98	83	285	149	86	87	233	152	89	91	267	157	93	98	306	187	103	93	294	194	100	129	284	216	105	137	259
0111000	242	270	246	253	318	315	268	290	338	309	329	301	342	278	283	315	383	2486	2352	2527	2976	2589	2477	2539	3038	2659	2755	2755	3199	2920	2743	2795	3261
0111000	243	313	172	202	788	329	185	214	837	362	179	220	781	347	194	196	842	3453	3311	3480	3990	3657	3447	3490	4196	3779	3735	3863	4665	4017	3721	4216	4556
0111000	244	45	19	46	91	60	23	24	64	21	23	23	87	23	22	23	94	2175	2106	2218	2737	2294	2243	2171	2717	2357	2342	2348	2837	2616	2368	2591	3019
0111001	245	216	242	223	285	229	259	268	332	241	288	271	315	248	281	254	295	277	299	323	378	288	284	288	370	351	341	301	353	309	303	324	414
0111001	246	84	80	86	128	88	84	86	108	96	89	95	111	99	97	129	143	109	103	137	131	118	108	161	161	117	125	113	152	122	141	140	137
0111001	247	20	20	20	22	22	20	20	26	20	19	20	23	22	21	21	27	404	217	228	676	364	281	268	682	405	266	254	690	447	244	258	651
0111010	248	344	353	349	451	370	388	367	455	352	403	379	464	400	401	397	487	714	741	762	816	771	736	790	928	797	863	848	846	878	903	853	925
0111010	249	514	351	415	899	577	409	462	949	574	420	436	954	564	428	436	962	2769	2680	2772	3363	2957	2787	2780	3454	3019	3063	2997	3380	3169	3241	3290	3537
0111010	250	393	261	302	785	444	274	315	826	468	325	318	858	453	324	331	879	2704	2551	2690	3183	2792	2684	2669	3341	2899	2881	2871	3255	3076	3063	3090	3360
0111011	251	11	12	13	14	12	10	14	14	14	11	14	15	13	12	14	16	229	103	141	521	251	105	108	529	283	211	115	564	291	117	128	535
0111011	252	74	47	48	52	50	49	51	53	51	50	53	55	55	58	54	64	363	170	190	620	339	202	204	597	333	196	191	636	343	193	266	620
0111011	253	43	43	45	45	45	44	45	51	45	47	48	53	48	48	49	51	302	172	187	619	313	233	179	610	328	213	200	625	341	226	217	627
0111100	254	36	39	63	59	43	50	70	43	42	69	43	43	52	47	47	49	50	45	49	51	100	52	51	54	48	50	53	49	54	56	66	64
0111100	255	52	52	54	52	55	55	55	109	82	58	78	66	59	61	61	61	64	70	69	92	66	65	67	66	75	71	71	83	72	73	74	80
0111100	256	271	249	282	283	262	264	296	273	305	307	309	335	315	353	319	321	303	342	384	358	325	316	345	354	325	398	361	338	364	367	384	452
0111101	257	71	72	74	72	76	77	78	80	79	81	82	82	82	80	83	96	84	88	96	93	91	105	90	98	92	93	101	116	95	98	99	99

continued on next page

Table A.3: *CitiesAuto* absolute (ms): average over all iterations of a query with DB intervals
(continued from previous page)

class	ID	+DB intervals																															
		+indirect keyword occurrences																−indirect keyword occurrences															
		+index intervals								−index intervals								+index intervals								−index intervals							
		+label index				−label index				+label index				−label index				+label index				−label index				+label index				−label index			
		ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG
		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0111101	258	8	8	9	7	7	7	9	8	7	8	10	9	9	8	9	8	8	8	10	33	8	8	9	9	9	8	9	13	18	8	10	10
0111110	259	540	517	505	515	551	525	538	546	571	613	563	618	619	596	642	645	633	627	705	714	657	655	667	741	664	728	725	838	708	816	750	866
0111110	260	13	11	12	20	12	12	33	17	12	15	33	18	12	13	14	21	13	13	24	22	13	14	14	24	14	14	19	25	17	16	15	27
0111110	261	293	285	263	281	301	322	280	323	297	292	295	344	309	359	365	361	318	318	370	417	362	360	368	374	372	416	362	395	405	400	398	446
0111111	262	12	12	13	14	13	11	13	16	14	13	13	14	13	13	14	14	13	13	16	17	21	14	16	15	13	15	15	18	14	18	16	16
0111111	263	5	5	6	6	13	9	10	14	6	6	7	7	12	9	12	13	6	6	8	6	22	10	11	14	7	8	8	7	13	17	13	14
0111111	264	8	9	9	8	8	8	9	9	9	9	10	9	9	17	10	10	10	9	11	10	9	10	11	10	10	12	13	10	10	11	12	48
0111111	265	8	7	8	7	14	12	13	14	7	8	9	8	14	19	13	15	8	9	10	8	17	13	12	18	8	10	12	9	30	15	14	18
1000000	266	429	237	268	1154	483	273	276	1224	497	283	282	1268	487	256	265	1269	4897	4685	5037	5852	5078	5137	4896	6046	5350	5391	5516	6220	5823	6139	5744	7020
1000000	267	556	315	293	1359	603	300	301	1364	634	309	322	1383	655	312	329	1408	2597	2453	2537	3241	2916	2591	2596	3361	2766	2740	2796	3810	3115	3088	3041	4651
1000000	268	350	176	231	900	362	206	218	908	410	193	244	917	359	204	192	889	354	182	195	900	418	218	228	855	413	249	233	854	445	299	256	841
1000001	269	430	261	221	941	487	214	220	984	495	252	229	962	491	256	234	1008	781	516	579	1332	892	567	556	1273	866	650	632	1269	946	657	662	1338
1000001	270	1160	548	549	2582	1235	603	602	2723	1268	590	609	2670	1326	605	653	2754	2678	1935	2066	4185	2832	2236	2022	4410	2829	2555	2350	4486	2989	2263	2265	4293
1000001	271	641	411	394	1638	668	375	414	1721	745	435	432	1758	725	458	437	1781	1335	1047	1173	2632	1372	1265	1075	2566	1464	1267	1230	2713	1502	1199	1206	2596
1000010	273	289	189	195	620	332	196	197	662	292	173	173	675	364	238	206	676	356	212	191	696	385	217	239	673	376	240	230	823	334	237	230	699
1000011	275	415	177	202	1067	442	187	212	1101	501	194	199	1069	514	229	204	1109	757	322	359	1626	796	405	359	1696	823	389	431	1653	888	432	425	1648
1000011	276	248	93	67	459	288	68	69	507	227	77	78	479	270	77	79	515	260	77	80	539	244	80	81	569	265	90	88	547	319	102	88	542
1000011	277	640	218	255	1436	669	256	268	1469	747	247	251	1486	760	255	284	1509	718	325	359	1484	757	390	352	1548	791	390	413	1544	919	456	400	1472
1000100	278	37	30	64	140	46	34	36	175	39	33	62	171	46	61	38	152	39	33	64	153	77	37	58	127	42	36	39	154	57	38	39	127
1000100	279	275	93	97	621	297	128	132	639	275	106	105	632	287	106	139	617	288	102	107	595	262	109	108	626	304	143	142	744	339	127	155	657
1000100	280	224	95	99	577	233	102	105	615	250	105	157	625	311	108	111	629	284	108	146	628	320	112	113	680	307	172	119	689	350	176	143	639
1000101	281	212	108	85	533	277	86	92	536	263	127	93	521	261	94	95	548	295	98	95	538	254	102	96	656	291	112	124	545	300	110	105	548
1000101	282	467	168	209	1020	472	201	233	1078	537	186	197	1056	499	216	231	1080	671	525	523	1122	720	523	580	1196	753	618	694	1200	791	640	623	1107
1000101	283	232	89	96	489	229	94	97	511	235	98	102	505	305	130	105	513	266	97	124	560	251	128	124	550	287	136	110	600	274	110	113	516
1000110	284	249	119	121	583	264	131	122	563	314	149	189	570	258	132	137	579	257	157	173	620	295	136	136	614	281	154	158	617	290	206	172	612
1000110	285	231	139	136	537	222	108	136	537	239	118	121	582	299	122	177	552	294	115	128	598	254	162	121	542	291	144	205	637	338	134	172	581
1000110	286	209	134	109	528	253	136	113	537	259	119	149	546	247	175	124	555	242	120	160	620	282	121	121	608	269	147	134	620	290	165	137	598
1000111	287	250	65	69	487	210	69	72	490	227	75	76	505	259	77	78	514	268	76	81	523	271	123	98	525	310	113	89	549	318	87	89	530

continued on next page

Table A.3: *CitiesAuto* absolute (ms): average over all iterations of a query with DB intervals
(continued from previous page)

class	ID	+DB intervals																															
		+indirect keyword occurrences																−indirect keyword occurrences															
		+index intervals								−index intervals								+index intervals								−index intervals							
		+label index				−label index				+label index				−label index				+label index				−label index				+label index				−label index			
		ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG
		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
1000111	288	197	67	72	508	208	68	74	505	273	77	75	503	269	79	78	509	233	75	83	548	269	91	80	564	255	109	96	534	296	99	116	543
1000111	289	195	70	74	457	262	71	76	522	235	103	80	495	257	80	111	516	257	104	89	511	250	93	85	545	278	98	120	562	324	108	94	538
1001000	290	339	202	181	740	336	212	216	788	398	192	223	786	410	250	205	777	1940	1769	1960	2381	2051	2009	1840	2432	2111	2126	2017	2449	2352	2140	2231	2497
1001000	291	238	124	181	635	280	156	154	679	295	133	136	634	269	132	172	692	2178	2121	2149	2688	2290	2199	2222	2944	2351	2319	2427	2829	2628	2323	2617	2826
1001000	292	305	191	168	861	365	201	175	854	382	199	214	918	398	181	220	932	4735	4621	4868	5462	5021	4966	4911	5668	5164	5511	5546	5800	5747	5519	5582	5837
1001001	293	309	190	170	639	313	195	236	653	357	211	184	700	394	251	201	666	336	196	240	673	384	205	208	750	391	208	240	729	455	225	253	744
1001001	294	217	121	142	516	282	116	119	526	245	123	154	532	250	126	129	548	550	430	433	831	609	445	449	925	601	480	526	1008	679	451	489	868
1001010	296	36	26	27	89	45	31	31	73	40	28	29	74	47	33	54	75	77	69	45	74	63	43	49	92	56	45	44	101	66	63	52	138
1001010	297	264	128	127	654	291	143	144	676	289	170	172	686	316	153	155	678	2496	2425	2543	3085	2645	2515	2539	3225	2721	2884	2705	3226	2866	2701	2781	3219
1001010	298	493	220	256	1186	493	281	258	1212	523	242	244	1190	557	247	252	1204	2513	2411	2569	3040	2675	2487	2496	3419	2741	2660	2804	3304	2915	3027	2796	3264
1001011	299	216	95	97	496	236	108	111	514	247	132	137	514	261	117	174	539	290	144	129	548	334	107	111	587	286	137	147	606	289	173	145	544
1001011	300	286	141	112	519	258	162	134	543	264	126	135	531	280	161	171	555	276	136	176	583	381	181	181	666	361	200	200	580	352	198	164	593
1001011	301	411	195	203	964	463	204	208	996	494	192	194	1018	506	226	202	1034	322	128	137	608	325	132	134	642	341	139	164	626	311	160	146	593
1001100	302	69	61	83	87	107	76	70	97	71	63	91	93	83	73	74	132	82	69	103	135	110	105	81	137	87	80	88	108	121	106	86	131
1001100	304	224	157	157	583	269	162	190	571	276	142	141	584	267	176	181	606	255	253	173	643	299	146	149	615	329	195	187	660	353	211	162	642
1001101	305	204	96	98	490	238	125	102	488	231	107	133	479	264	108	111	485	263	169	112	512	327	110	112	532	259	128	124	551	275	119	135	543
1001101	306	13	9	9	18	23	18	18	27	13	9	9	20	23	17	17	57	14	12	10	22	56	19	20	35	14	11	11	19	27	18	21	29
1001110	307	263	103	106	529	224	111	114	517	286	172	148	522	247	148	152	558	246	120	149	556	333	177	154	605	294	143	143	593	323	143	147	558
1001110	308	209	131	112	529	257	111	113	546	243	118	128	519	268	122	123	553	297	183	123	550	300	122	133	591	304	137	200	601	305	183	146	557
1001110	309	262	106	78	537	247	81	83	513	291	87	88	544	273	89	117	562	271	91	124	598	319	89	89	622	289	93	99	584	314	98	127	581
1001111	310	196	81	85	477	232	85	113	463	224	93	94	477	266	96	98	504	264	122	103	539	264	94	98	549	255	111	122	553	335	109	110	529
1001111	311	224	98	129	481	212	104	106	477	253	111	113	471	234	165	117	516	235	132	158	549	287	118	120	550	278	134	132	552	316	132	131	545
1001111	312	16	12	14	33	23	20	21	40	17	35	16	61	25	21	22	40	18	17	18	36	26	22	23	43	20	17	20	34	29	22	32	46
1010000	313	10	7	8	59	11	7	8	103	12	7	8	55	12	7	9	48	18	12	10	46	22	59	12	108	48	12	13	49	23	11	11	44
1010000	314	290	181	181	696	302	198	166	690	326	205	172	672	333	180	174	707	2723	2426	2571	3776	2966	2482	2539	4055	3032	2724	2890	4101	3219	2833	2863	3909
1010000	315	362	218	221	784	326	250	261	804	370	230	257	796	403	238	237	819	4691	4352	4569	6091	5056	4508	4544	6369	5141	4992	4990	6523	5581	5295	5504	6342
1010001	316	10	8	10	13	40	8	10	13	11	10	11	13	11	10	11	14	10	8	12	14	13	9	12	16	12	11	10	15	12	10	13	15
1010001	317	11	10	10	12	10	10	10	12	11	9	11	15	11	29	11	14	10	9	13	19	13	10	10	16	12	10	12	14	12	11	12	15

continued on next page

Table A.3: *CitiesAuto* absolute (ms): average over all iterations of a query with DB intervals
(continued from previous page)

class	ID	+DB intervals																															
		+indirect keyword occurrences																−indirect keyword occurrences															
		+index intervals								−index intervals								+index intervals								−index intervals							
		+label index				−label index				+label index				−label index				+label index				−label index				+label index				−label index			
		ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG
		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
1010001	318	215	94	97	486	219	99	101	472	260	130	105	487	247	107	110	514	239	137	113	537	325	108	110	583	318	135	116	560	300	133	129	542
1010010	319	13	43	15	84	14	15	15	137	14	15	15	138	16	15	17	111	23	19	19	157	23	18	18	159	23	20	20	161	32	20	20	155
1010010	320	209	138	137	535	223	106	140	541	239	114	116	544	267	116	143	581	241	115	145	599	255	147	153	725	272	154	172	662	309	182	164	588
1010010	321	2515	2312	2476	2869	2663	2560	2621	3005	2780	2677	2719	3146	2908	2705	2862	3219	3010	2924	2998	3538	3357	2984	3023	3598	3218	3190	3272	3702	3590	3395	3379	3720
1010011	322	10	10	11	13	12	9	11	13	13	9	12	14	11	10	13	16	11	14	10	13	17	10	11	16	13	11	10	17	13	10	11	16
1010011	323	23	23	24	26	45	23	24	47	25	24	26	58	26	25	27	54	271	146	124	537	258	122	126	544	272	131	157	580	279	135	137	513
1010011	324	10	9	10	13	10	9	10	14	11	9	10	14	11	10	10	14	12	12	10	15	12	10	12	15	12	23	11	18	14	10	13	15
1010100	325	10	7	9	25	11	8	10	25	10	8	9	25	12	39	10	33	10	9	11	27	10	8	9	27	12	14	9	27	32	9	11	26
1010100	326	250	100	145	536	272	134	162	577	279	141	142	555	287	133	115	566	284	148	115	580	273	141	144	596	333	132	161	591	286	152	157	599
1010100	327	251	124	105	582	264	103	108	588	276	139	112	607	284	116	142	604	284	177	116	658	311	137	123	630	287	132	129	662	343	151	132	633
1010101	328	238	82	84	494	251	86	89	512	236	91	94	513	273	94	95	519	243	145	94	587	252	93	95	529	291	100	101	560	328	102	103	547
1010101	329	245	78	83	510	226	82	84	510	293	94	91	524	272	97	94	524	271	148	102	535	279	118	93	524	268	101	101	548	339	99	101	553
1010101	330	10	7	10	14	31	8	9	19	11	8	10	16	11	10	9	17	11	9	10	17	10	9	9	18	12	8	10	46	12	8	10	18
1010110	331	253	149	125	603	269	163	128	576	261	134	134	573	260	192	136	589	262	185	137	621	346	147	141	597	284	171	187	616	379	148	205	607
1010110	332	59	48	74	205	61	50	71	207	67	53	54	209	68	54	55	210	73	74	57	232	74	107	82	243	72	64	62	209	81	89	61	250
1010110	333	236	126	139	540	259	160	142	517	242	138	144	555	278	117	148	530	275	162	122	623	324	124	155	591	304	137	141	599	353	155	140	582
1010111	334	9	7	9	12	10	8	9	13	12	9	9	13	11	7	9	13	10	9	9	15	14	8	9	15	11	8	14	16	13	12	10	14
1010111	335	33	28	31	91	34	30	32	65	36	31	34	66	38	32	34	66	38	41	63	71	46	35	36	104	43	43	40	75	69	43	38	71
1010111	336	9	7	9	13	8	8	10	10	9	8	9	10	9	9	10	11	11	11	11	11	11	9	11	12	10	10	14	15	10	9	11	12
1011000	337	19	19	20	62	20	20	22	63	22	21	21	63	21	22	22	74	2192	2199	2156	2617	2554	2187	2169	2769	2361	2535	2361	2835	2594	2365	2447	2827
1011000	338	191	192	196	264	209	212	214	256	233	212	245	281	243	282	232	269	2389	2455	2477	2999	2780	2383	2482	3134	2621	2737	2753	3267	2913	2609	2738	3087
1011000	339	16	12	14	46	20	12	13	52	18	12	14	65	24	14	14	48	53	16	17	42	22	16	17	39	24	19	19	43	27	26	19	43
1011001	340	285	147	156	553	320	182	156	550	289	185	192	588	288	166	196	587	705	610	618	1018	738	620	601	1132	773	652	662	1084	785	670	683	1037
1011001	341	12	9	10	18	14	9	11	43	16	10	11	19	11	9	11	19	425	346	298	696	444	283	328	721	470	348	315	739	496	396	311	721
1011001	342	14	18	11	17	14	14	12	17	15	11	13	15	13	12	12	15	365	247	259	651	353	239	218	651	415	308	262	646	409	263	237	667
1011010	343	20	72	28	120	15	18	20	95	21	18	22	65	19	22	21	65	59	20	22	88	38	24	25	156	29	25	26	83	32	29	32	113
1011010	344	16	9	10	40	22	16	14	47	15	10	13	41	20	16	15	47	24	12	14	49	30	17	18	59	29	17	20	76	34	19	19	57
1011010	345	18	17	18	66	24	29	26	71	20	20	17	67	25	30	27	71	30	61	22	122	37	29	31	95	31	29	63	112	37	34	64	113

continued on next page

Table A.3: *CitiesAuto* absolute (ms): average over all iterations of a query with DB intervals
(continued from previous page)

class	ID	+DB intervals																															
		+indirect keyword occurrences																−indirect keyword occurrences															
		+index intervals								−index intervals								+index intervals								−index intervals							
		+label index				−label index				+label index				−label index				+label index				−label index				+label index				−label index			
		ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG
		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
1011011	346	19	19	19	20	21	22	23	23	51	20	22	23	22	59	22	23	239	115	123	548	303	127	120	526	269	149	130	576	274	129	155	545
1011011	347	8	7	8	10	8	7	8	10	29	7	9	13	9	7	9	11	11	8	10	11	9	8	10	11	10	8	9	12	12	9	12	12
1011011	348	8	7	8	10	8	8	8	41	9	8	9	13	11	8	9	11	263	140	104	539	238	100	128	546	301	140	125	570	286	107	115	522
1011100	349	319	175	182	628	317	213	186	638	299	201	229	665	345	231	263	635	309	214	264	702	350	205	203	821	363	240	243	775	356	217	272	695
1011100	350	11	9	12	24	10	10	11	54	10	10	13	55	10	10	13	26	10	11	13	79	11	10	11	85	11	10	21	28	10	11	14	26
1011100	351	9	11	11	55	10	10	11	25	10	10	11	26	10	11	12	32	10	12	12	27	11	10	12	60	12	12	11	29	11	11	12	26
1011101	352	1	0	2	2	1	1	2	22	2	1	1	3	1	0	2	2	1	0	1	2	3	0	1	9	2	0	2	2	2	0	3	2
1011101	353	26	25	52	28	27	26	32	36	57	47	29	30	30	28	30	32	34	33	30	35	29	29	30	47	35	33	62	36	32	32	34	36
1011110	354	339	339	350	385	357	388	347	386	370	346	407	378	407	386	397	415	373	407	450	482	456	416	430	542	435	467	453	476	465	447	579	505
1011110	355	2	3	1	6	2	2	2	8	4	2	2	6	2	2	2	9	2	4	2	7	3	2	2	10	2	2	2	5	2	2	35	6
1011111	356	1295	1304	1	1302	1344	1349	1	1420	1393	1415	1	1439	1472	1462	1	1501	1548	1830	1	1673	1578	1574	2	1789	1694	1878	2	1767	1770	1903	2	1753
1011111	357	9	7	8	10	9	7	8	11	10	9	10	10	9	8	9	11	11	9	9	11	11	8	9	14	10	9	9	13	10	9	11	32
1011111	358	1561	1573	1598	1623	1659	1686	1663	1719	1700	1716	1765	1785	1777	1812	1838	1852	1860	2308	1997	1944	1946	1978	1955	2146	2027	2129	2245	2112	2119	2087	2282	2114
1100000	359	195	74	2	445	232	75	2	482	222	92	1	489	256	88	1	506	231	97	1	529	268	87	2	494	311	99	2	519	276	96	2	527
1100000	360	17	11	1	154	22	19	1	157	18	13	2	155	24	18	1	159	19	18	2	162	24	22	2	179	22	14	1	164	28	22	1	160
1100000	361	256	79	1	472	214	84	1	465	231	113	2	489	282	94	1	494	241	111	2	548	305	139	2	555	267	112	2	554	348	135	2	500
1100001	362	234	92	2	564	300	97	1	536	261	100	2	567	301	104	2	582	561	447	3	899	694	437	2	977	682	492	2	872	670	462	2	917
1100001	363	230	114	107	488	218	98	97	483	240	94	98	507	240	123	136	536	240	97	109	557	294	99	128	587	336	104	144	560	295	131	135	535
1100001	364	17	14	2	64	24	17	2	95	22	15	1	64	26	17	2	71	29	18	2	119	44	21	2	136	38	18	32	87	42	22	2	93
1100010	365	209	106	108	558	245	116	107	541	235	118	139	559	270	121	138	562	244	116	127	592	260	122	125	618	334	185	154	610	321	147	175	600
1100010	366	237	77	80	539	229	82	80	547	238	88	85	549	275	90	89	536	271	85	94	598	285	111	90	623	273	99	112	607	283	112	99	610
1100010	367	210	80	2	556	272	107	2	569	266	86	2	561	267	88	1	527	269	83	2	573	328	87	2	640	341	115	2	562	331	105	2	588
1100011	368	203	93	1	477	215	98	2	458	232	106	2	490	238	108	1	444	263	124	1	527	300	116	2	533	275	197	2	516	300	178	2	532
1100011	369	248	91	94	512	267	99	102	515	260	104	103	529	299	110	112	549	268	129	138	591	287	120	157	616	331	141	170	564	341	141	135	554
1100011	370	10	8	1	13	11	8	2	40	11	9	2	16	11	10	1	15	11	8	2	15	18	9	2	18	13	9	2	18	13	11	2	15
1100100	371	210	105	140	518	250	114	178	532	265	119	120	537	245	133	153	528	296	180	128	588	265	125	156	618	268	165	169	576	284	147	161	603
1100100	372	260	104	133	590	272	137	109	598	252	176	170	625	283	148	118	609	258	115	157	677	350	115	119	660	284	147	157	682	289	140	185	614
1100100	373	223	109	113	539	266	110	113	549	278	114	120	561	283	147	153	572	309	121	127	607	382	127	175	596	354	155	139	597	335	156	138	592

continued on next page

Table A.3: *CitiesAuto* absolute (ms): average over all iterations of a query with DB intervals
(continued from previous page)

class	ID	+DB intervals																															
		+indirect keyword occurrences																−indirect keyword occurrences															
		+index intervals								−index intervals								+index intervals								−index intervals							
		+label index				−label index				+label index				−label index				+label index				−label index				+label index				−label index			
		ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG
		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
1100101	374	22	15	47	42	33	21	51	50	25	17	20	44	33	21	23	51	26	19	19	79	38	22	24	59	27	23	41	46	71	25	44	53
1100101	375	231	107	103	496	244	104	109	508	262	111	113	451	240	114	117	514	238	114	148	551	246	143	118	522	261	143	138	543	288	218	214	549
1100101	376	217	97	1	516	230	127	1	528	249	138	1	530	252	112	1	552	275	137	1	618	313	133	2	604	323	152	2	594	331	131	2	575
1100110	377	235	105	109	561	244	105	137	541	290	123	148	517	242	147	122	553	281	123	154	588	261	123	125	608	294	155	133	589	281	141	147	617
1100110	378	220	130	135	531	229	108	114	542	237	118	146	579	248	147	185	556	271	135	128	629	281	118	154	588	288	145	143	593	292	136	150	581
1100110	379	289	130	110	549	279	140	110	539	294	147	121	543	275	123	124	555	274	178	128	579	283	119	152	657	304	139	182	603	310	136	162	582
1100111	380	20	10	11	35	29	16	20	45	23	11	13	40	30	18	19	45	23	35	14	39	33	19	21	52	26	12	14	75	38	18	20	48
1100111	381	17	7	1	34	25	11	1	42	20	8	1	37	27	11	1	44	20	8	1	37	31	10	1	48	21	9	1	40	29	11	1	54
1100111	382	399	158	3	960	447	171	1	1002	485	208	2	1001	496	218	1	993	481	207	3	1043	506	227	2	1152	589	240	2	1125	601	260	2	1085
1101000	383	3	4	1	4	10	10	1	11	4	3	1	4	10	10	1	11	43	7	2	77	78	14	1	60	48	7	2	51	56	13	2	58
1101000	384	12	11	1	47	20	19	3	54	13	13	2	48	20	19	1	53	31	16	2	104	30	21	1	109	25	18	2	71	33	24	2	84
1101000	385	16	11	1	152	22	21	1	128	19	12	2	183	24	27	1	159	18	13	2	162	26	21	2	161	20	15	1	193	30	20	2	161
1101001	386	16	12	14	30	55	14	17	38	21	13	14	32	28	15	18	64	25	19	19	38	35	51	24	44	27	20	22	38	35	23	32	44
1101001	387	20	3	4	60	28	10	12	64	23	3	4	30	28	10	11	68	34	6	6	32	30	10	14	37	27	4	6	32	38	11	15	67
1101001	388	245	116	1	504	232	94	1	544	276	102	2	578	255	103	3	558	289	110	2	612	269	105	2	602	351	130	2	636	338	137	2	596
1101010	389	23	12	2	48	40	15	2	83	28	12	2	50	35	14	2	65	116	120	3	189	155	109	3	192	141	116	2	196	152	167	2	196
1101010	390	2	2	2	25	7	8	1	14	2	2	2	9	7	7	1	13	2	2	3	9	8	8	1	15	2	2	2	8	7	8	2	42
1101010	391	269	152	126	639	280	157	187	651	265	139	136	684	324	140	142	660	2470	2420	2442	3007	2593	2481	2534	3200	2792	2832	2796	3174	3032	2830	2849	3164
1101011	392	3	3	4	5	12	10	12	40	4	3	4	5	11	10	10	12	5	3	6	13	12	10	13	16	5	11	5	6	12	10	12	13
1101011	393	8	9	9	11	16	15	16	28	9	9	10	11	15	15	16	47	282	125	169	530	242	114	116	552	310	116	150	513	295	155	126	530
1101011	394	17	11	11	66	19	11	13	29	18	11	12	30	20	12	14	32	23	16	16	36	23	15	15	38	23	15	20	36	25	17	17	37
1101100	395	6098	6184	6315	6403	6614	6601	6641	6860	6901	6899	7004	7084	7133	7228	7338	7352	7551	7759	8138	8135	8765	7899	8020	8594	8439	9160	8941	9091	8647	9121	8832	8727
1101100	396	13	18	18	18	22	21	20	19	17	18	14	17	20	19	21	25	19	16	19	18	23	41	22	20	22	19	15	20	21	20	22	21
1101100	397	0	1	0	1	10	9	10	11	2	0	1	0	9	9	9	10	1	0	1	0	10	10	9	10	0	1	0	0	10	9	10	9
1101101	398	1	2	3	2	37	8	10	9	1	1	3	2	7	8	8	9	1	2	2	2	38	8	9	8	2	2	3	2	37	8	9	28
1101101	399	1	2	3	2	2	2	3	3	2	2	3	2	2	2	3	2	1	2	3	3	2	2	2	4	2	2	3	2	1	2	3	2
1101110	400	1	1	3	9	9	6	7	14	1	2	3	37	8	5	11	14	1	2	3	8	9	6	6	15	2	2	3	9	8	5	6	14
1101110	401	5	1	2	11	10	34	7	17	5	0	2	11	10	5	34	16	5	0	2	43	11	6	8	37	7	1	2	10	10	5	36	18

continued on next page

Table A.3: *CitiesAuto* absolute (ms): average over all iterations of a query with DB intervals
(continued from previous page)

class	ID	+DB intervals																															
		+indirect keyword occurrences																−indirect keyword occurrences															
		+index intervals								−index intervals								+index intervals								−index intervals							
		+label index				−label index				+label index				−label index				+label index				−label index				+label index				−label index			
		ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG
		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
1101110	402	12	0	2	48	19	5	6	81	15	0	2	54	20	4	5	54	15	0	2	52	22	4	6	63	16	1	2	55	22	4	5	56
1101111	403	252	75	1	470	210	101	1	464	252	86	1	512	233	88	2	505	232	91	2	541	267	124	1	524	258	131	2	566	291	126	1	523
1101111	404	11	2	6	16	16	6	2	22	12	3	2	21	24	5	1	23	13	2	2	22	17	6	1	25	16	3	1	25	30	6	1	25
1101111	405	1	1	1	2	9	6	8	8	2	1	2	2	8	6	7	8	2	1	2	2	18	7	8	9	2	0	2	3	8	9	16	8
1110000	406	9	7	9	23	11	8	9	24	10	8	9	25	10	8	10	54	11	8	10	59	11	8	10	29	10	8	9	58	11	9	9	54
1110000	407	16	13	13	43	15	13	14	98	17	13	15	52	16	14	15	45	22	16	18	43	23	17	17	72	23	48	38	45	24	17	18	43
1110000	408	151	154	150	276	226	156	159	264	170	162	166	237	204	194	174	298	2333	2265	2403	2890	2823	2344	2334	3181	2644	2504	2594	3041	2718	2720	2579	3087
1110001	409	357	185	2	786	350	164	2	851	438	204	2	832	386	184	1	857	629	430	2	1120	658	413	2	1196	699	447	2	1144	729	516	2	1139
1110001	410	5	5	2	8	15	9	2	17	5	6	2	10	14	11	2	16	5	6	3	8	15	8	2	17	6	6	2	10	14	14	2	17
1110001	411	69	50	54	235	89	52	84	184	77	55	57	194	75	85	60	242	428	336	257	863	457	292	266	904	477	410	313	851	504	325	316	851
1110010	412	3039	3055	3141	3285	3262	3309	3286	3517	3347	3394	3420	3612	3547	3522	3590	3786	3851	3801	4046	4426	4329	3956	3989	4741	4300	4246	4298	4743	4561	4685	4268	4657
1110010	413	937	905	922	1264	990	984	958	1333	1009	988	1029	1301	1053	1061	1052	1340	1140	1141	1141	1492	1289	1129	1129	1575	1236	1206	1256	1650	1412	1227	1274	1608
1110010	414	873	905	951	1243	961	952	962	1271	1032	1003	1057	1330	1053	1063	1049	1341	1148	1127	1113	1524	1403	1299	1131	1571	1302	1233	1321	1570	1318	1283	1212	1614
1110011	415	85	53	53	76	63	77	88	76	91	88	62	77	68	59	64	84	315	184	224	621	349	198	228	605	348	211	250	671	395	207	241	646
1110011	416	9	7	9	11	9	7	10	11	10	7	9	12	10	9	9	11	9	7	9	11	11	41	9	12	10	14	10	32	11	9	10	12
1110011	417	21	4	5	25	22	4	5	26	25	4	5	27	24	4	6	28	91	5	6	50	46	4	5	50	49	7	6	50	49	4	6	50
1110100	418	9	7	9	24	10	8	9	25	10	8	9	24	11	9	9	27	10	7	9	27	10	8	9	26	12	8	12	28	11	9	11	27
1110100	419	9	8	9	55	10	8	9	24	10	8	10	26	10	8	10	25	10	8	9	26	12	13	9	28	12	8	11	27	11	8	10	27
1110100	420	9	7	11	19	9	8	10	20	10	8	9	20	10	8	11	19	9	8	10	22	12	54	9	23	10	10	11	21	13	8	19	52
1110101	421	10	9	9	12	9	8	10	13	9	8	10	13	11	9	10	13	9	8	9	14	12	9	8	14	12	8	12	45	12	9	9	13
1110101	422	19	4	5	25	21	4	5	27	22	4	5	26	22	4	5	27	22	4	6	30	28	4	5	30	28	4	5	30	31	4	5	30
1110101	423	10	9	10	10	10	9	10	12	9	10	9	11	10	8	10	14	10	9	9	13	13	46	9	14	10	8	10	12	13	8	10	14
1110110	424	199	94	99	520	239	111	113	529	263	134	112	535	263	142	169	546	236	123	151	582	276	138	146	575	262	160	124	573	339	133	128	578
1110110	425	61	52	49	201	73	50	51	203	67	53	57	179	66	80	57	211	67	57	71	221	73	62	56	254	129	60	64	194	77	82	63	241
1110110	426	78	50	50	206	61	52	53	185	92	55	55	219	69	56	55	181	69	54	61	222	75	59	59	216	74	60	67	203	76	90	62	216
1110111	427	11	8	9	65	12	8	9	65	12	9	11	94	12	10	11	65	38	10	12	128	13	11	11	70	15	11	14	81	14	12	13	99
1110111	428	12	10	11	34	13	9	11	63	15	10	13	68	15	10	12	60	220	87	121	531	258	95	117	504	264	108	137	645	317	104	112	508
1110111	429	13	9	9	25	14	9	10	25	16	10	10	26	15	9	10	55	16	9	12	27	16	10	12	27	19	35	11	31	19	11	11	31

continued on next page

Table A.3: *CitiesAuto* absolute (ms): average over all iterations of a query with DB intervals
(continued from previous page)

class	ID	+DB intervals																															
		+indirect keyword occurrences																−indirect keyword occurrences															
		+index intervals								−index intervals								+index intervals								−index intervals							
		+label index				−label index				+label index				−label index				+label index				−label index				+label index				−label index			
		ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG
		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
1111000	430	74	100	99	179	79	72	73	155	78	85	110	130	85	82	84	135	104	133	105	166	108	160	104	165	132	105	106	181	112	104	107	175
1111000	431	11	7	10	23	11	8	10	24	10	8	10	25	11	9	9	53	10	8	11	29	11	8	10	56	12	8	10	28	11	8	10	26
1111000	432	1	1	0	1	0	1	0	0	0	0	0	1	0	1	1	1	0	1	0	2	0	0	1	0	1	0	1	0	1	1	0	1
1111001	433	12	10	11	99	13	11	11	100	12	12	13	72	12	41	12	71	20	13	16	153	20	13	16	157	20	14	16	128	25	14	17	148
1111001	434	1	1	2	0	0	1	2	0	0	1	2	0	1	0	2	1	1	1	2	0	0	31	2	1	0	1	2	0	1	0	2	0
1111001	435	10	10	11	16	11	30	12	16	11	10	11	16	11	13	13	44	404	223	231	654	390	239	231	666	384	242	251	661	466	273	279	659
1111010	436	13	9	10	37	13	9	9	38	14	8	11	38	13	8	9	39	18	12	12	41	19	11	12	41	20	11	14	41	21	12	14	41
1111010	437	352	201	199	972	368	201	264	989	422	274	219	999	407	242	251	1001	4412	4228	4420	5427	4916	5026	4425	5705	4970	4706	5056	5544	5141	5017	4805	5707
1111010	438	11	6	9	33	12	6	10	34	11	7	10	33	41	7	38	34	21	12	10	40	18	10	10	40	21	12	14	44	22	11	11	35
1111011	439	58	59	90	70	63	61	63	73	65	67	92	102	70	67	77	80	81	83	78	123	86	74	108	104	91	104	87	129	89	81	81	101
1111011	440	13	12	12	77	13	13	13	49	14	12	14	77	14	12	15	78	24	16	19	73	47	17	20	77	28	18	20	73	26	17	19	120
1111011	441	11	8	9	12	11	9	9	12	11	9	10	14	10	9	10	15	273	201	150	565	285	177	152	601	336	159	204	563	344	189	166	604
1111100	442	1	0	1	0	0	1	0	1	1	0	0	0	1	1	0	1	0	1	1	0	0	1	1	1	1	0	1	0	0	1	0	1
1111100	443	0	1	0	1	1	0	1	0	0	0	0	0	1	0	1	0	0	1	1	0	1	0	1	0	1	0	0	0	0	1	0	2
1111100	444	3	4	4	2	2	3	3	3	2	2	4	2	3	3	2	2	4	2	2	2	3	4	3	34	3	3	3	2	4	4	3	2
1111101	445	2	2	2	2	2	2	2	2	2	2	3	2	2	2	3	2	2	2	3	3	2	3	3	2	1	3	3	2	1	2	3	2
1111101	446	2	1	3	2	2	2	3	2	2	2	3	2	1	2	2	2	2	2	3	2	2	2	23	2	2	2	3	3	3	2	3	2
1111110	447	3	1	2	9	2	0	2	8	2	1	2	10	3	1	2	8	3	0	3	8	3	1	2	8	3	0	2	65	3	0	2	10
1111110	448	2	2	2	8	2	2	3	7	2	1	3	5	2	2	3	8	2	2	3	8	1	2	3	11	2	2	3	5	2	2	3	7
1111111	449	1	0	1	2	2	1	2	2	2	1	2	2	2	0	2	2	2	0	2	2	2	0	7	2	2	0	2	3	2	0	2	2
1111111	450	1	0	1	2	1	0	2	2	2	1	1	2	2	0	2	2	2	1	2	3	2	1	2	2	2	1	2	2	2	0	2	2
1111111	451	1	1	1	0	1	0	2	1	0	1	1	0	0	0	1	1	0	0	2	0	0	1	1	1	1	1	2	0	1	1	2	0

Table A.4: *CitiesAuto* absolute (ms): average over all iterations of a query without DB intervals

class	ID	−DB intervals																															
		+indirect keyword occurrences																−indirect keyword occurrences															
		+index intervals								−index intervals								+index intervals								−index intervals							
		+label index				−label index				+label index				−label index				+label index				−label index				+label index				−label index			
		ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG
		32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
0000000	101	540	442	420	812	577	414	418	855	606	418	481	848	614	494	442	890	3701	3600	3601	3867	3575	3478	3434	3884	4076	3917	3988	4299	3823	3860	3801	3945
0000000	102	3348	2845	2834	3930	3439	2908	2925	4130	3705	3102	3179	4193	3726	3217	3229	4342	7118	6623	6656	7421	7355	6804	6801	8002	7778	7191	7375	7872	7782	7379	7497	8048
0000001	104	666	385	333	767	672	366	369	810	721	386	365	812	736	395	428	865	771	587	614	820	750	624	630	875	799	654	593	883	831	697	654	849
0000001	105	4789	4128	4210	5096	4957	4343	4379	5497	5356	4612	4659	5593	5456	4790	4770	5773	5481	4878	4909	5789	5640	5001	5036	6174	6008	5311	5360	6134	6061	6111	5637	6241
0000010	108	445	433	441	482	500	458	466	505	464	483	489	535	508	498	503	569	501	493	525	549	567	509	512	613	584	551	535	652	577	647	559	603
0000011	109	2262	1902	1918	2362	2305	1993	1957	2573	2490	2100	2124	2648	2596	2197	2215	2671	2711	2203	2233	2879	1732	1229	1239	1934	3007	2377	2515	3090	1843	1487	1451	1943
0000100	110	265	269	256	278	320	269	270	275	297	298	296	274	294	288	290	324	293	288	324	297	340	301	328	378	337	339	329	307	326	365	364	349
0000101	111	858	722	724	943	883	737	773	1019	983	816	795	1014	973	836	822	1026	1462	1169	1176	1573	1523	1168	1176	1660	1671	1256	1239	1689	1621	1351	1346	1713
0000111	112	447	305	321	469	443	289	292	524	495	318	349	507	508	344	346	527	510	320	324	528	496	329	356	607	548	389	371	601	557	354	412	588
0001000	113	1104	978	960	1282	1202	1023	1008	1409	1261	1063	1093	1382	1282	1155	1117	1469	2252	2135	2119	2364	1365	1188	1188	1523	2458	2256	2362	2593	1480	1258	1440	1515
0001001	114	656	505	502	758	691	542	580	821	729	556	591	839	761	578	612	820	957	771	807	1091	994	828	802	1076	1028	898	813	1168	1025	884	954	1093
0001001	115	330	189	217	424	402	205	235	452	359	202	206	420	403	274	256	519	664	544	519	722	721	555	560	778	719	545	572	798	725	555	585	762
0001011	118	433	280	264	493	455	299	279	519	482	286	327	527	477	311	304	535	477	300	333	520	343	138	148	402	567	359	346	579	392	144	151	397
0001100	119	459	472	481	474	470	539	536	527	514	523	547	497	542	572	553	580	515	516	545	532	569	557	577	591	595	564	576	582	612	625	670	647
0001100	121	19	20	20	16	27	31	30	32	21	24	22	18	28	33	54	27	24	22	22	19	70	32	33	60	24	24	23	20	34	30	34	27
0001100	122	75	103	80	79	105	88	87	123	85	86	87	86	94	149	96	93	115	124	90	91	99	101	100	108	97	97	104	95	103	105	115	102
0001101	123	705	549	534	786	758	591	626	848	771	590	616	845	810	644	649	873	796	630	637	848	826	684	752	918	940	680	738	885	938	704	821	924
0001101	124	65	56	87	64	78	70	72	84	94	87	64	71	83	76	77	115	71	65	66	75	88	79	81	93	85	91	82	103	91	80	92	94
0001110	125	5731	5829	5900	5922	5992	6044	6091	6409	6459	6511	6594	6640	6665	6713	6791	6830	6820	6868	6926	6978	7018	7103	7084	7475	7484	7497	7895	7513	7478	7542	7900	7537
0001110	127	13323	13435	13629	13774	13932	13986	14159	14998	15058	15124	22402	15410	15458	15590	15766	15844	15860	15954	16104	16255	16335	16471	16560	17337	17443	17412	17000	17760	17382	17534	18108	17627
0001111	128	602	572	617	654	635	639	678	676	665	697	1185	687	696	694	729	736	707	710	711	753	736	767	738	845	752	819	809	770	770	804	901	784
0001111	129	11	31	43	19	18	19	19	28	12	13	22	19	22	20	20	27	13	15	16	21	22	21	22	26	13	13	16	22	20	21	22	48
0001111	130	591	606	609	613	636	629	636	681	661	673	1160	707	692	700	700	706	697	729	744	711	729	727	773	778	792	766	742	790	828	831	864	806
0010000	131	86	81	111	144	88	82	94	151	94	88	159	126	122	120	94	150	3510	3214	3243	3775	3601	3327	3328	4011	3869	3520	3413	4171	3842	3536	3827	4052
0010000	132	1243	1279	1330	1338	1354	1329	1348	1405	1424	1388	2436	1453	1453	1456	1425	1556	5058	4776	4807	5261	5211	4884	4928	5598	5517	5189	5044	5967	5579	5548	5462	5679
0010000	133	2920	2993	3070	3079	3085	3127	3145	3263	3235	3300	4953	3392	3409	3405	3442	3545	4594	4442	4421	4736	4671	4578	4612	5055	4982	4823	4726	5187	4971	5091	4891	5111
0010001	134	9711	9641	9885	10178	10216	10120	10206	10965	10876	10823	11856	11251	11290	11132	11223	11499	12066	11778	11842	12408	12352	12066	12184	13194	13196	12823	12476	13379	13131	13450	13377	13369
0010001	136	4832	4898	4940	5066	5055	5045	5069	5386	5427	5481	5482	5606	5591	5615	5667	5754	6420	6078	6119	6581	6538	6220	6336	7019	7008	6658	6449	6926	6985	7044	7069	7087

continued on next page

Table A.4: *CitiesAuto* absolute (ms): average over all iterations of a query without DB intervals
(continued from previous page)

class	ID	−DB intervals																															
		+indirect keyword occurrences																−indirect keyword occurrences															
		+index intervals								−index intervals								+index intervals								−index intervals							
		+label index				−label index				+label index				−label index				+label index				−label index				+label index				−label index			
		ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG
		32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
0010010	138	9938	9931	9969	10506	10389	10198	10298	11300	11168	11045	11054	11588	11526	11404	11466	12022	14055	13743	13801	14431	14421	14085	14222	15331	15328	14978	14527	16492	15263	15826	14937	15545
0010011	140	384	203	204	490	395	207	214	505	426	232	251	513	436	238	311	546	1967	934	980	2254	1998	1007	973	2325	2205	1064	996	2417	2165	1047	1042	2375
0010100	141	169	145	145	156	173	191	176	166	182	159	162	206	168	190	167	176	196	191	193	210	166	166	197	216	172	196	208	190	179	181	181	193
0010100	142	174	194	199	187	179	203	191	206	194	215	194	205	199	195	197	241	200	197	227	239	245	236	206	230	239	232	212	254	283	267	240	232
0010100	143	110	112	147	148	112	114	117	157	120	122	150	165	127	125	130	136	152	122	124	140	148	130	161	144	134	134	133	146	136	143	134	150
0010101	144	2670	2562	2596	2735	2762	2654	2652	2983	2964	2806	2865	2998	3008	2907	2914	3119	3067	2913	2963	3197	3130	3069	2998	3424	3396	3192	3132	3755	3346	3167	3212	3473
0010110	145	503	517	518	539	494	529	509	564	554	564	593	585	577	578	606	622	645	624	589	601	656	604	648	642	659	647	610	645	629	699	626	652
0010110	146	352	298	301	349	364	339	335	398	401	351	360	406	432	365	347	425	383	368	372	455	393	375	392	463	434	371	368	448	445	424	430	462
0010110	147	14	9	11	18	35	13	14	27	16	10	11	19	23	13	15	25	16	10	12	23	48	14	15	69	17	11	13	21	30	15	15	30
0010111	148	308	351	348	325	345	351	351	375	342	369	375	385	377	354	384	369	382	359	427	400	369	399	399	433	428	422	426	484	396	415	424	438
0010111	149	223	204	206	223	211	236	213	222	265	225	229	235	232	268	237	234	247	266	241	239	309	274	248	259	284	279	279	253	286	272	271	259
0010111	150	303	251	259	289	313	259	289	350	330	314	308	346	345	288	316	359	360	302	322	362	336	310	369	353	372	312	312	357	379	318	378	397
0011000	151	2779	2798	2934	2876	2889	2957	2933	3133	3105	3136	3164	3276	3228	3212	3283	3330	3281	3334	3331	3410	3397	3434	3412	3618	3599	3603	3544	3616	3667	3650	3792	3722
0011000	152	1598	1525	1635	1698	1663	1607	1585	1751	1768	1705	1728	1796	1818	1783	1765	1894	4226	3970	4041	4385	4295	4135	4164	4686	4636	4347	4260	4669	4575	4313	4609	4751
0011000	153	1137	1120	1185	1142	1136	1182	1140	1279	1267	1239	1268	1290	1245	1314	1285	1349	3611	3504	3532	3816	3703	3578	3598	3994	3988	3779	3690	4025	3999	3781	3989	4092
0011001	155	536	565	652	551	609	523	568	582	589	589	594	596	602	577	611	614	1012	865	881	1081	1063	886	866	1175	1115	911	927	1151	1125	917	914	1188
0011001	156	4382	4371	4648	4483	4514	4524	4483	4841	4784	4870	4876	4865	4873	4916	4967	5045	5116	5066	5122	5186	5176	5213	5306	5482	5496	5508	5381	5686	5437	5439	5635	5555
0011010	157	383	202	274	508	392	235	237	544	384	220	248	545	393	227	229	563	1979	1881	1837	2104	2038	1869	1858	2252	2156	2033	1925	2192	2155	2000	2034	2264
0011011	158	3953	4026	4508	4122	4137	4107	4183	4376	4392	4440	4440	4595	4561	4540	4597	4638	4969	4859	4856	5128	5091	4954	4984	5381	5374	5218	5072	5383	5391	5289	5494	5468
0011011	159	221	195	200	226	237	235	205	241	217	218	245	222	223	222	226	228	568	387	384	606	601	413	417	652	580	468	401	634	592	440	488	650
0011011	160	24	24	25	25	26	24	25	27	27	25	26	29	28	26	27	28	338	165	167	349	317	142	146	362	342	193	155	362	340	159	169	393
0011100	161	497	421	459	427	432	459	463	484	486	490	490	535	481	509	505	505	509	543	544	547	528	563	532	517	549	573	519	551	583	576	551	579
0011100	162	430	425	433	464	436	483	510	553	518	511	519	498	477	479	484	552	552	523	519	535	525	524	563	604	613	581	572	548	556	536	627	592
0011100	163	133	135	138	161	176	144	145	149	144	153	147	149	182	158	183	157	153	158	162	178	214	165	167	166	205	169	163	189	200	206	207	173
0011101	164	4198	4207	4286	4316	4346	4374	4444	4563	4556	4627	4660	4696	4727	4778	4795	4792	4855	4843	4907	4930	4964	5009	5038	5218	5233	5229	5118	5202	5235	5168	5484	5258
0011101	165	11	9	10	14	12	10	11	69	12	19	11	15	12	10	14	47	14	12	13	13	15	9	11	13	13	10	11	13	12	10	14	14
0011110	166	387	419	393	438	457	403	411	435	462	454	486	441	438	499	467	452	506	470	458	518	528	490	494	558	490	582	528	519	523	478	541	535
0011110	167	161	97	94	104	140	106	124	144	109	102	104	121	144	106	107	123	111	106	139	119	124	111	117	149	118	114	114	131	129	143	120	159

continued on next page

Table A.4: *CitiesAuto* absolute (ms): average over all iterations of a query without DB intervals
(continued from previous page)

class	ID	−DB intervals																															
		+indirect keyword occurrences																−indirect keyword occurrences															
		+index intervals								−index intervals								+index intervals								−index intervals							
		+label index				−label index				+label index				−label index				+label index				−label index				+label index				−label index			
		ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG
		32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
0011110	168	536	580	554	554	555	559	591	634	600	566	595	609	639	607	652	663	606	623	627	638	638	637	673	675	688	661	658	686	706	678	747	688
0011111	169	83	80	83	88	98	110	88	103	118	96	98	97	103	96	96	109	123	91	94	123	135	123	103	139	119	103	107	107	113	104	109	141
0011111	170	81	77	89	85	84	80	81	93	89	86	88	95	93	113	89	97	94	91	91	98	107	100	128	102	125	100	98	103	128	98	116	109
0011111	171	542	533	578	577	553	559	556	588	591	589	594	612	590	614	634	619	624	619	625	672	637	632	641	693	709	664	654	670	671	674	687	682
0100000	174	1415	1423	1440	1519	1483	1474	1494	1614	1591	1586	1626	1677	1642	1659	1677	1725	1750	1758	1755	1800	83	65	69	120	1901	1907	1830	1933	87	78	69	94
0100001	175	426	362	367	388	413	369	348	413	442	389	365	423	425	435	378	458	182	208	216	184	218	215	192	197	220	190	218	194	229	229	200	227
0100001	176	1270	1125	1127	1397	1362	1199	1209	1505	1469	1277	1255	1527	1518	1290	1316	1589	1799	1614	1616	1877	1845	1697	1695	1979	1929	1760	1699	1979	1927	1758	1752	1997
0100001	177	8869	8781	8969	9294	9241	9084	9176	10048	9957	9749	9884	10324	10271	10090	10119	10607	10967	10634	10711	11261	1017	1355	1421	1154	11942	12048	11334	11990	1102	1425	1436	1175
0100010	178	551	343	377	675	566	360	390	742	598	375	404	716	586	421	451	736	2144	2045	2021	2248	1989	1793	1810	2193	2339	2173	2106	2496	2097	1971	1978	2150
0100010	179	1114	1035	1048	1318	1178	1031	1073	1412	1280	1106	1142	1437	1280	1156	1165	1432	2925	2705	2717	3048	2940	2778	2795	3239	3171	2967	2915	3208	3112	3021	3044	3259
0100010	180	3510	3479	3543	3582	3609	3634	3666	3866	3852	3897	3924	3965	4064	4046	4042	4156	4092	4116	4198	4234	4234	4269	4316	4507	4478	4808	4385	4544	4552	4563	4544	4575
0100011	181	395	366	363	394	405	406	408	461	436	396	398	464	447	449	440	453	179	270	275	183	200	308	310	198	203	359	314	194	205	298	325	205
0100011	182	52	55	53	54	59	60	59	71	57	56	68	59	67	90	66	68	59	59	61	34	42	69	71	41	35	72	66	36	43	77	74	50
0100011	183	285	169	142	355	308	182	187	381	345	154	185	376	380	166	175	404	328	208	164	392	373	151	152	386	398	176	174	407	380	184	160	425
0100100	184	113	84	86	98	126	93	94	114	128	91	93	110	110	97	101	119	113	126	123	111	114	100	103	135	124	133	129	121	120	157	136	128
0100100	185	113	111	116	114	153	151	135	132	124	126	129	126	145	138	162	138	138	132	133	137	144	171	172	195	150	152	142	143	179	167	153	154
0100100	186	601	549	578	595	591	600	571	674	657	629	634	678	654	651	656	670	666	672	678	683	697	695	691	776	768	729	712	752	729	723	785	750
0100101	187	71	42	44	116	75	48	68	95	99	47	47	95	85	59	53	152	81	69	50	126	84	54	55	102	86	49	54	102	91	52	77	104
0100101	188	265	243	275	255	316	255	262	302	275	271	281	283	288	282	316	297	340	285	342	298	333	323	327	362	316	329	329	350	348	323	330	356
0100101	189	316	179	183	385	375	219	215	403	374	230	224	384	384	228	224	422	356	199	202	406	363	202	205	442	390	235	241	436	410	214	263	455
0100110	190	30	16	17	35	43	24	24	73	33	18	18	37	39	23	25	43	35	38	18	39	42	27	27	45	35	18	20	70	41	25	28	45
0100110	191	560	526	517	615	548	542	552	611	609	578	578	592	604	626	599	665	607	644	611	692	670	631	632	702	637	754	642	683	675	689	747	719
0100111	192	31	25	28	42	38	30	31	52	33	27	29	40	41	32	33	51	34	30	30	46	63	33	34	55	39	36	31	47	43	42	36	53
0100111	193	158	121	133	139	174	164	167	156	150	140	138	155	164	148	205	191	165	145	145	161	166	211	187	200	182	182	156	201	202	181	193	207
0100111	194	25	10	12	29	34	17	18	36	29	11	13	30	39	17	18	36	31	11	13	32	34	19	19	37	37	16	14	59	36	19	19	47
0101000	196	497	283	310	640	454	327	361	681	540	346	338	670	533	322	327	671	1938	1800	1761	2057	2017	1828	1814	2186	2170	1986	1883	2176	2109	2008	1931	2178
0101000	197	470	292	317	599	458	338	308	635	498	337	314	673	494	353	357	685	1478	1313	1327	1561	1540	1386	1452	1659	1590	1452	1419	1698	1653	1514	1504	1700
0101001	198	363	194	198	427	374	207	235	504	366	209	212	453	384	222	249	444	679	523	525	747	715	530	683	757	827	562	546	760	726	593	563	816

continued on next page

Table A.4: *CitiesAuto* absolute (ms): average over all iterations of a query without DB intervals
(continued from previous page)

class	ID	−DB intervals																															
		+indirect keyword occurrences																−indirect keyword occurrences															
		+index intervals								−index intervals								+index intervals								−index intervals							
		+label index				−label index				+label index				−label index				+label index				−label index				+label index				−label index			
		ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG
		32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
0101001	199	314	164	167	397	340	194	171	392	344	202	188	424	407	244	183	406	666	507	510	697	650	489	529	803	708	535	486	754	715	505	559	766
0101001	200	321	173	176	391	343	185	156	381	349	161	161	418	354	166	176	419	623	484	509	690	665	498	507	717	688	507	476	727	706	493	518	733
0101010	202	548	394	408	723	592	455	395	739	598	416	426	791	655	453	435	789	2245	2053	2013	2330	1979	1770	1843	2127	2493	2267	2177	2499	2068	2000	1955	2180
0101011	204	1197	1031	1047	1231	1209	1050	1068	1356	1323	1149	1130	1378	1366	1217	1144	1422	1359	1215	1192	1457	344	132	137	399	1465	1288	1256	1532	409	157	172	370
0101011	205	164	141	147	157	209	145	178	170	192	160	191	200	175	162	190	178	207	173	173	212	28	19	19	36	191	191	184	241	31	40	20	43
0101011	206	376	234	230	463	414	250	244	512	410	249	259	482	427	262	267	463	430	288	293	502	327	150	153	408	529	314	336	521	340	167	187	398
0101100	207	2	3	3	1	9	8	9	6	2	2	2	0	8	9	8	7	2	12	2	1	9	9	8	6	2	2	3	0	8	9	8	7
0101100	208	7	26	7	7	15	16	15	14	7	7	7	6	14	15	15	13	7	7	8	7	14	16	45	14	8	8	8	7	16	27	15	14
0101100	209	14	15	21	12	17	18	18	17	15	33	14	13	17	17	18	46	13	13	14	12	19	18	19	18	15	14	16	21	18	29	19	17
0101101	210	3	0	2	5	10	6	8	11	5	0	2	5	10	6	7	11	4	1	2	5	11	7	9	13	4	0	2	6	10	6	7	12
0101101	211	2	3	3	2	8	9	38	8	1	2	3	1	7	8	9	8	2	2	3	2	8	8	9	8	2	2	3	2	7	9	9	7
0101101	212	296	127	131	348	340	131	157	354	314	169	141	334	336	142	168	340	313	166	141	375	306	141	143	378	352	146	148	387	358	166	151	392
0101101	213	2	3	3	1	9	8	9	9	2	3	3	2	7	28	9	8	2	2	3	2	8	9	9	8	2	2	3	2	8	9	9	8
0101110	214	21	13	12	27	25	17	15	33	22	13	14	26	36	15	17	34	23	12	13	29	29	16	18	67	24	12	13	32	30	19	16	37
0101110	215	17	12	13	20	19	17	16	26	18	12	13	23	19	17	25	55	19	12	15	21	21	17	21	26	18	12	14	21	20	16	17	24
0101110	216	18	18	19	19	26	46	27	29	19	20	21	41	26	28	28	29	20	20	22	41	29	27	29	31	21	21	23	24	29	28	30	30
0101111	217	71	66	66	79	80	69	73	90	101	73	128	87	113	77	77	93	109	76	79	91	92	136	112	101	94	105	82	97	118	100	107	102
0101111	218	10	8	9	11	13	12	12	16	11	7	9	12	15	12	13	15	11	8	9	12	15	13	14	15	13	8	9	13	16	13	14	18
0101111	219	3	4	5	2	10	10	10	10	3	3	4	3	10	38	10	9	3	3	4	4	10	10	12	11	3	3	5	3	9	9	11	10
0110000	220	3972	3828	3913	4228	4088	3950	4028	4582	4410	4346	4343	4689	4565	4395	4516	4838	6040	5558	5568	6285	6128	5678	6417	6736	6593	6272	5880	6678	6539	6647	6297	6790
0110000	221	3981	3904	4020	4130	4096	4055	4125	4391	4388	4342	4367	4458	4495	4505	4420	4613	6688	6367	6418	6923	6785	6550	6675	7371	7223	7288	6721	7310	7220	7280	7030	7412
0110000	222	599	553	568	656	612	567	595	692	614	590	628	698	660	640	607	715	2090	1755	1762	2217	2131	1804	1805	2327	2256	1936	1875	2343	2252	1947	1902	2396
0110001	223	663	667	650	735	686	657	666	780	749	713	710	796	754	721	758	840	806	753	759	894	854	804	783	948	867	834	806	949	881	864	857	1008
0110001	225	2344	2359	2400	2407	2441	2394	2489	2618	2623	2626	2651	2706	2673	2687	2718	2773	3145	2984	3032	3271	3228	3075	3126	3489	3462	3290	3177	3493	3415	3376	3384	3517
0110010	226	383	237	212	498	347	212	217	553	390	198	200	525	379	202	229	528	1956	1796	1812	2053	2034	1850	1830	2192	2197	1882	1907	2224	2179	2119	1944	2260
0110010	227	2566	2581	2661	2730	2680	2645	2725	2929	2937	2901	2925	2984	2983	2995	3005	3084	3257	3166	3168	3393	3391	3228	3249	3647	3623	3444	3354	3626	3641	3513	3470	3730
0110010	228	2566	2593	2663	2665	2646	2674	2672	2905	2901	2898	2947	2984	3001	3004	3035	3112	3276	3140	3166	3425	3383	3258	3325	3593	3620	3480	3379	3634	3598	3771	3654	3680
0110011	229	1298	1310	1334	1378	1387	1358	1375	1487	1465	1458	1475	1531	1533	1493	1514	1592	1573	1543	1555	1606	1610	1586	1623	1672	1673	1775	1669	1676	1709	1885	1857	1737

continued on next page

Table A.4: *CitiesAuto* absolute (ms): average over all iterations of a query without DB intervals
(continued from previous page)

class	ID	−DB intervals																															
		+indirect keyword occurrences																−indirect keyword occurrences															
		+index intervals								−index intervals								+index intervals								−index intervals							
		+label index				−label index				+label index				−label index				+label index				−label index				+label index				−label index			
		ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG
		32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
0110100	230	136	109	110	117	111	111	111	125	144	116	150	150	125	183	123	132	122	153	123	134	125	180	125	140	134	131	131	142	136	133	133	159
0110100	231	45	39	64	38	40	41	40	41	44	68	65	42	45	43	53	43	44	45	45	44	46	48	47	45	46	49	47	47	47	50	49	56
0110100	232	92	83	87	103	111	91	94	100	93	93	95	98	94	94	96	104	95	96	97	107	97	98	100	116	130	100	102	113	103	108	104	114
0110101	233	177	202	182	183	185	183	186	220	195	194	197	199	226	203	227	228	203	227	206	222	263	207	244	223	273	239	240	244	273	220	222	224
0110101	234	32	31	32	34	33	32	32	35	36	33	35	37	36	36	35	42	39	34	38	37	39	43	39	39	42	38	38	41	42	42	47	41
0110101	235	9	11	10	9	10	9	10	10	9	12	11	10	10	10	11	10	30	11	11	11	11	12	12	11	11	11	13	12	11	13	13	12
0110110	236	296	291	304	309	346	332	342	342	328	329	363	306	341	310	372	318	321	370	374	351	351	355	328	402	373	411	340	373	340	438	389	404
0110110	237	952	850	872	957	950	846	916	996	1046	925	959	1056	1042	962	968	1071	1053	981	1009	1126	1110	998	1003	1205	1120	1135	1026	1168	1154	1136	1137	1199
0110110	238	648	677	687	692	667	663	697	735	740	704	741	760	767	751	745	774	742	731	769	780	788	738	832	861	828	895	807	857	830	895	841	894
0110111	239	32	34	33	32	33	32	33	35	35	34	35	35	37	35	45	38	68	35	36	37	39	63	39	38	41	40	40	40	39	41	39	40
0110111	240	56	70	55	59	58	54	56	64	84	58	60	66	65	59	60	67	66	60	61	93	67	61	126	70	87	112	65	86	72	67	66	73
0110111	241	153	79	78	193	178	80	80	179	168	86	86	183	171	87	88	210	171	88	113	189	200	87	104	222	186	102	95	225	211	116	120	227
0111000	242	220	197	256	249	226	234	205	237	222	271	217	288	247	219	247	238	2586	2467	2454	2727	2652	2526	2513	2890	2825	2813	2584	2848	2823	2767	2698	2895
0111000	243	391	187	195	614	416	204	210	607	365	200	235	645	445	240	245	659	2621	2439	2493	2714	2634	2480	2503	2845	2854	2657	2557	2899	2845	2796	2708	2936
0111000	244	31	32	29	47	27	27	29	48	28	30	30	49	32	62	31	49	2343	2204	2260	2522	2413	2292	2297	2578	2670	2380	2404	2714	2569	2455	2566	2701
0111001	245	478	511	556	568	555	533	564	576	592	579	554	614	576	586	573	624	645	608	609	634	621	626	624	651	642	642	659	661	655	665	654	673
0111001	246	166	154	157	166	163	159	162	203	210	196	175	202	180	177	204	194	189	183	210	274	195	216	239	254	231	200	195	243	205	199	198	219
0111001	247	34	31	33	60	36	32	35	38	39	37	35	37	36	36	37	39	450	278	283	492	469	283	302	536	513	320	270	545	490	300	308	551
0111010	248	6789	6869	7024	7090	7116	7147	7204	7663	7659	7673	7740	7900	7854	7953	7934	8042	8281	8348	8421	8514	8537	8609	9355	9097	9085	8831	8886	9046	9102	9318	9243	9183
0111010	249	3412	3332	3372	3703	3547	3411	3454	3990	3820	3708	3756	4128	3955	3839	3839	4191	5628	5523	5566	5799	5767	5606	6013	6218	6165	5963	5819	6150	6156	6039	6345	6215
0111010	250	2240	2152	2199	2513	2353	2243	2274	2673	2565	2390	2391	2726	2660	2448	2456	2762	4270	4096	4159	4358	4342	4210	4350	4709	4606	4339	4346	4629	4639	4611	4685	4724
0111011	251	27	18	18	20	19	23	19	23	23	19	20	27	21	19	21	43	329	136	163	367	336	139	143	355	353	144	142	411	331	173	145	399
0111011	252	60	53	55	56	57	53	56	60	60	58	59	62	61	60	61	63	396	202	204	449	410	261	247	460	435	239	241	495	408	248	238	467
0111011	253	116	119	127	123	152	118	120	132	131	126	128	131	134	133	132	134	484	305	308	516	467	286	295	539	508	295	299	540	524	330	345	542
0111100	254	37	37	52	39	44	44	44	47	42	67	42	42	49	48	47	48	49	44	48	45	77	51	51	51	49	46	58	49	52	53	60	52
0111100	255	47	41	49	42	40	40	40	72	42	44	45	44	46	44	51	44	46	46	53	44	46	46	47	45	48	48	67	79	51	49	60	81
0111100	256	192	186	191	232	218	222	227	211	206	209	239	263	264	221	222	239	218	249	220	249	221	248	226	227	231	228	229	257	231	244	242	272
0111101	257	98	73	75	75	75	77	79	80	100	80	116	90	82	108	84	85	83	93	106	86	115	87	91	88	113	88	91	91	93	117	114	92

continued on next page

Table A.4: *CitiesAuto* absolute (ms): average over all iterations of a query without DB intervals
(continued from previous page)

class	ID	−DB intervals																															
		+indirect keyword occurrences																−indirect keyword occurrences															
		+index intervals								−index intervals								+index intervals								−index intervals							
		+label index				−label index				+label index				−label index				+label index				−label index				+label index				−label index			
		ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG
		32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
0111101	258	5	5	7	5	6	6	15	6	6	6	7	6	7	6	7	6	6	6	14	6	6	7	8	8	7	7	8	7	7	6	9	6
0111110	259	397	378	394	403	472	412	445	465	421	443	443	473	482	424	449	474	426	457	465	470	464	467	495	455	500	475	510	518	527	493	533	524
0111110	260	9	9	10	11	9	10	11	12	10	11	13	13	10	11	12	12	12	10	12	14	12	11	12	13	11	12	14	13	12	11	25	13
0111110	261	287	307	314	301	293	293	318	371	345	340	330	323	324	328	406	363	387	340	338	340	375	400	441	375	387	387	390	389	390	399	419	424
0111111	262	11	10	11	10	9	11	11	11	10	10	13	12	10	12	13	11	10	11	12	12	10	12	13	12	10	11	12	12	11	20	15	12
0111111	263	5	5	6	5	12	10	11	12	5	6	8	5	12	8	11	13	7	6	8	6	13	10	12	13	6	6	7	6	14	10	12	23
0111111	264	7	6	8	7	7	7	8	7	7	7	10	7	7	7	8	8	8	8	9	7	7	8	9	8	8	7	9	7	8	9	10	8
0111111	265	7	7	9	8	15	10	12	15	9	7	11	37	44	11	12	40	8	8	9	8	16	13	12	16	9	8	11	8	15	31	13	16
1000000	266	518	232	240	860	530	297	274	875	591	253	321	883	569	317	317	898	3711	3451	3449	3882	3782	3579	3740	4181	4037	3609	3666	4141	4075	3956	3840	4256
1000000	267	712	328	306	929	688	334	338	972	759	355	375	979	746	346	325	991	2056	1843	1829	2178	2109	1889	1929	2366	2215	1954	1947	2316	2225	2214	2102	2338
1000000	268	420	202	265	591	426	198	201	622	450	230	209	631	488	210	212	644	450	235	256	588	496	205	209	603	465	241	243	613	487	252	269	553
1000001	269	578	313	320	695	627	349	378	726	687	356	366	745	672	369	380	761	855	601	577	959	874	592	603	1003	958	610	592	1019	949	672	622	1000
1000001	270	6337	5569	5693	6816	6546	5765	5822	7354	7036	6179	6222	7493	7219	6387	6414	7677	6027	5287	5314	6279	6133	5445	6028	6643	6567	5581	5659	6698	6511	5710	5819	6752
1000001	271	4221	3935	4250	4558	4401	4061	4107	4888	4773	4404	4376	5010	4908	4502	4508	5159	5215	4908	4901	5538	5373	4992	5252	5818	5669	5086	5278	5842	5704	5488	5522	5898
1000010	273	10475	10386	10574	10863	10877	10793	10903	11716	11717	11609	11720	12069	12092	11960	12012	12392	12377	12255	12388	12702	12694	12623	13208	13583	13524	13180	13407	13482	13581	13621	14129	13746
1000011	275	597	282	291	730	645	256	256	773	702	285	305	753	679	283	309	759	1033	450	453	1141	987	458	470	1166	1105	476	487	1160	1121	532	491	1180
1000011	276	300	118	116	311	308	91	91	329	294	97	97	357	304	97	100	347	299	97	101	368	305	100	118	406	359	103	106	365	327	116	141	392
1000011	277	836	307	293	965	857	296	299	1072	955	316	355	1055	947	351	358	1068	886	427	416	1049	933	425	414	1061	1035	442	432	1053	971	483	487	1040
1000100	278	252	181	210	243	268	216	194	261	210	200	233	232	251	217	211	283	245	236	214	243	231	249	257	327	233	220	250	313	271	231	242	331
1000100	279	283	111	112	437	288	114	141	417	334	145	122	398	316	124	126	423	350	119	161	459	342	148	126	437	375	150	160	476	338	135	161	449
1000100	280	337	141	116	404	341	114	119	393	317	151	131	452	366	154	151	420	339	151	149	429	343	153	126	466	386	127	129	419	397	171	158	445
1000101	281	278	130	110	375	306	108	109	361	305	114	143	365	334	119	119	355	359	112	115	378	314	115	183	392	331	119	138	376	367	119	121	382
1000101	282	585	219	244	701	609	215	244	722	640	253	230	768	681	230	234	745	647	478	470	708	681	482	483	756	665	489	497	733	689	519	544	739
1000101	283	304	114	116	338	312	117	119	397	310	124	150	360	311	124	153	368	309	122	154	370	311	125	135	383	362	167	131	397	392	144	152	425
1000110	284	307	196	141	354	340	144	175	399	366	152	165	431	354	210	194	415	346	182	154	385	356	210	201	466	370	186	163	402	343	165	217	435
1000110	285	270	127	157	369	317	136	135	356	301	140	143	362	361	143	146	393	329	142	177	382	354	144	177	427	356	151	180	441	331	152	171	419
1000110	286	329	128	133	399	280	131	161	396	356	142	170	396	309	154	170	396	345	143	173	399	312	145	149	397	362	150	187	422	357	153	199	429
1000111	287	265	84	119	308	297	85	89	348	296	92	98	376	328	123	100	360	300	122	101	369	358	99	99	373	367	137	101	408	338	113	137	383

continued on next page

Table A.4: *CitiesAuto* absolute (ms): average over all iterations of a query without DB intervals
(continued from previous page)

class	ID	−DB intervals																															
		+indirect keyword occurrences																−indirect keyword occurrences															
		+index intervals								−index intervals								+index intervals								−index intervals							
		+label index				−label index				+label index				−label index				+label index				−label index				+label index				−label index			
		ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG
		32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
1000111	288	265	89	91	344	272	90	98	348	301	97	100	329	308	98	102	364	324	128	101	373	313	155	100	430	344	108	105	356	324	107	106	361
1000111	289	264	94	96	335	297	121	95	362	317	103	134	370	301	104	105	362	301	105	132	342	305	107	177	378	358	109	114	391	378	111	127	360
1001000	290	605	350	360	732	565	380	361	739	630	378	381	764	641	426	410	788	1690	1510	1566	1782	1763	1563	1643	1899	1840	1725	1622	1952	1842	1776	1765	1957
1001000	291	294	140	172	463	300	172	175	484	322	150	190	461	350	182	187	492	1737	1541	1526	1823	1750	1578	1635	1919	1893	1635	1622	1927	1877	1697	1727	1960
1001000	292	824	647	686	1171	881	668	712	1240	911	743	721	1245	989	762	736	1275	4092	3943	4011	4299	4194	4048	4254	4576	4522	4231	4154	4541	4483	4582	4394	4617
1001001	293	386	189	203	435	399	206	204	467	447	217	240	462	434	219	254	489	400	237	214	517	440	222	264	476	465	263	224	508	444	258	266	524
1001001	294	282	203	143	394	316	175	144	408	333	178	188	387	316	155	190	366	536	350	385	579	547	422	364	629	584	419	395	627	616	404	404	586
1001010	296	387	385	413	409	406	381	394	475	437	392	395	448	446	413	409	441	494	430	450	472	451	461	482	495	503	470	481	512	478	458	480	500
1001010	297	527	317	347	640	549	389	339	684	537	375	352	686	559	422	400	726	2139	1957	1972	2214	2204	2024	2091	2353	2305	2265	2071	2374	2335	2292	2195	2407
1001010	298	581	268	310	842	592	279	324	865	699	351	354	860	682	318	323	885	1997	1780	1781	2066	1993	1848	1928	2181	2152	1916	1875	2183	2108	2094	1958	2190
1001011	299	440	226	256	434	461	273	255	493	457	252	254	503	476	268	314	485	477	292	303	519	320	131	145	406	487	286	337	520	340	139	138	401
1001011	300	390	245	256	457	468	242	272	487	448	270	272	482	477	291	279	488	474	261	259	550	488	305	315	556	522	307	295	499	489	331	323	574
1001011	301	573	259	255	631	588	230	230	714	643	248	251	706	648	253	256	713	330	174	177	396	397	190	157	413	386	164	185	414	390	195	162	416
1001100	302	111	120	136	116	122	109	137	161	119	114	169	128	157	169	119	139	130	119	117	135	140	126	134	163	140	134	136	145	144	142	167	155
1001100	304	337	155	217	400	324	186	190	392	310	161	202	423	341	166	171	459	311	167	167	425	344	193	201	494	360	253	201	446	417	205	209	481
1001101	305	271	123	127	313	278	125	126	328	354	139	136	359	316	137	145	341	335	158	134	371	336	135	142	382	396	150	167	368	336	154	145	395
1001101	306	15	10	12	16	23	19	19	27	17	10	12	18	26	18	19	27	17	11	13	18	28	18	20	29	21	13	12	22	28	19	21	58
1001110	307	299	161	136	379	314	132	170	392	301	141	145	386	329	174	149	397	308	141	152	402	313	144	185	446	363	154	154	421	370	168	180	429
1001110	308	280	155	130	370	280	194	135	411	308	140	142	364	308	151	174	404	333	170	144	371	365	171	181	393	326	177	151	414	334	195	156	420
1001110	309	295	100	103	346	308	104	105	355	325	135	112	387	307	113	112	445	330	108	113	437	310	113	134	413	349	157	148	386	356	119	151	393
1001111	310	264	108	118	331	294	110	110	383	293	133	147	350	363	122	147	333	306	143	146	362	330	123	142	371	358	150	132	352	323	132	159	391
1001111	311	264	129	145	369	282	134	161	348	321	142	146	360	304	172	149	334	326	144	175	368	314	185	151	392	323	185	155	378	325	186	168	412
1001111	312	21	17	17	24	29	23	25	33	24	17	20	25	30	24	26	41	53	17	20	26	59	30	48	66	25	19	21	28	32	44	26	33
1010000	313	14	8	9	31	13	29	11	31	14	8	10	31	35	9	11	32	21	11	12	30	22	11	18	30	25	14	13	31	25	11	14	30
1010000	314	4024	3908	3983	4262	4163	4033	4026	4572	4520	4326	4396	4676	4644	4492	4512	4809	7390	6945	6982	7720	7573	7127	7512	8216	8026	7679	7613	8177	8069	7687	7546	8314
1010000	315	1278	1076	1138	1427	1309	1091	1156	1476	1397	1213	1258	1565	1428	1251	1260	1569	6104	5582	5635	6515	6282	5767	6066	6869	6630	6052	6208	6838	6665	6493	6152	6985
1010001	316	33	28	31	51	31	29	31	33	34	35	33	36	35	32	35	36	35	32	34	39	35	33	48	37	40	37	49	64	37	35	39	39
1010001	317	32	28	31	31	32	30	63	34	33	34	33	35	36	32	40	42	60	32	34	38	35	33	50	37	40	36	43	37	36	55	38	39

continued on next page

Table A.4: *CitiesAuto* absolute (ms): average over all iterations of a query without DB intervals (continued from previous page)

class	ID	−DB intervals																															
		+indirect keyword occurrences																−indirect keyword occurrences															
		+index intervals								−index intervals								+index intervals								−index intervals							
		+label index				−label index				+label index				−label index				+label index				−label index				+label index				−label index			
		ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG
		32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
1010001	318	298	120	123	323	307	149	125	330	326	130	133	359	331	132	147	363	303	128	135	346	334	153	194	356	355	147	148	360	355	169	142	389
1010010	319	17	14	16	65	15	14	15	37	17	15	17	66	19	16	18	67	25	19	19	76	24	19	31	49	27	20	48	49	56	19	20	86
1010010	320	300	183	130	348	286	160	135	364	325	171	144	387	356	142	170	369	332	195	168	399	310	144	199	420	387	176	152	427	352	157	181	417
1010010	321	2055	1829	1865	2163	2061	1889	1927	2269	2166	2072	2047	2314	2275	2093	2090	2396	2319	2116	2132	2439	2386	2160	2218	2584	2472	2360	2283	2563	2481	2488	2310	2595
1010011	322	16	16	16	18	16	16	16	19	17	16	18	19	18	16	18	20	18	16	18	22	20	15	18	24	19	21	20	20	20	17	19	23
1010011	323	36	38	54	39	38	68	38	41	41	40	42	40	41	51	42	42	350	164	163	371	331	164	171	425	366	208	211	404	353	173	176	385
1010011	324	16	12	23	16	16	12	13	17	16	13	14	18	16	13	16	18	17	13	16	18	16	13	17	18	19	17	16	19	26	15	40	25
1010100	325	12	9	14	15	11	10	11	15	12	11	12	15	14	10	11	16	11	9	11	66	12	10	12	16	14	12	11	16	12	10	13	48
1010100	326	343	152	125	372	329	126	128	363	341	130	133	397	385	134	135	437	374	130	132	380	324	187	137	422	398	149	138	421	371	166	172	409
1010100	327	281	118	149	402	288	121	148	415	337	159	130	428	317	131	135	406	314	155	143	458	345	129	137	480	382	161	137	484	370	136	140	456
1010101	328	291	105	108	319	311	107	110	360	329	114	115	394	313	113	144	348	335	125	115	386	344	116	118	368	353	161	148	368	366	119	125	399
1010101	329	334	101	121	349	285	133	103	362	342	110	128	344	314	134	115	384	399	106	109	366	326	120	115	370	385	113	114	389	346	135	145	408
1010101	330	12	9	16	14	13	9	20	15	13	11	11	15	13	11	11	15	13	10	12	38	13	10	11	16	15	10	11	16	14	11	11	46
1010110	331	282	146	144	412	288	141	146	400	312	154	185	415	318	186	194	441	317	185	211	418	348	190	168	471	362	160	171	468	400	213	173	433
1010110	332	67	113	54	102	71	69	76	105	79	92	72	105	98	64	66	107	79	64	60	136	77	65	76	110	90	76	68	113	83	72	71	142
1010110	333	308	130	132	408	305	162	134	387	302	140	144	393	338	144	172	424	335	141	143	401	351	148	168	459	362	177	207	414	366	171	176	422
1010111	334	12	9	11	15	12	9	10	15	13	9	11	16	14	35	11	15	14	11	12	15	13	9	12	15	14	10	11	16	15	11	12	24
1010111	335	52	45	49	61	54	47	49	69	57	49	52	65	58	51	55	63	58	51	53	68	65	54	55	78	61	55	56	67	64	60	57	71
1010111	336	11	10	11	11	11	10	11	12	12	18	12	12	12	18	12	21	11	11	12	34	12	11	12	15	13	12	13	13	12	43	14	13
1011000	337	19	19	20	38	19	19	21	48	20	20	27	40	21	20	21	39	2360	2199	2242	2474	2446	2323	2330	2606	2655	2508	2387	2665	2585	2393	2414	2744
1011000	338	274	265	254	318	281	258	283	289	326	263	298	314	277	307	336	354	2644	2478	2508	2788	2688	2557	2748	2971	2875	2630	2698	2914	2863	2765	2698	2975
1011000	339	21	15	16	34	21	15	16	35	25	16	18	36	23	16	18	36	27	18	21	30	25	23	21	29	31	24	22	31	28	19	41	31
1011001	340	1537	1355	1366	1612	1586	1393	1423	1716	1644	1524	1508	1761	1727	1563	1566	1804	2046	1903	1879	2116	2056	1910	1988	2277	2155	2122	2014	2270	2270	2162	2086	2240
1011001	341	13	10	12	14	13	10	12	16	14	10	13	14	12	11	12	16	486	303	333	532	549	340	336	553	500	315	319	555	529	323	328	563
1011001	342	44	43	42	46	45	48	45	48	50	45	47	50	51	47	49	52	446	278	272	490	447	271	303	528	510	306	309	531	477	339	314	536
1011010	343	18	17	21	36	19	19	18	40	20	20	22	35	20	20	23	35	81	24	26	72	33	25	26	46	36	31	27	78	40	26	27	77
1011010	344	16	11	12	28	24	15	42	37	18	13	14	29	50	16	17	35	36	15	17	36	43	19	21	45	36	16	17	37	47	19	20	45
1011010	345	18	18	20	34	23	23	27	72	19	20	21	37	26	33	28	46	32	24	25	44	37	51	34	52	37	25	27	76	37	32	30	79

continued on next page

Table A.4: *CitiesAuto* absolute (ms): average over all iterations of a query without DB intervals
(continued from previous page)

class	ID	−DB intervals																															
		+indirect keyword occurrences																−indirect keyword occurrences															
		+index intervals								−index intervals								+index intervals								−index intervals							
		+label index				−label index				+label index				−label index				+label index				−label index				+label index				−label index			
		ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG
		32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
1011011	346	71	97	73	82	74	73	75	79	79	78	80	80	80	81	83	119	365	203	205	430	402	207	224	419	424	213	216	421	424	227	220	451
1011011	347	12	9	10	11	10	9	11	12	11	9	12	41	12	11	20	14	12	9	11	13	13	9	11	14	12	10	11	15	12	9	12	13
1011011	348	12	9	10	11	12	9	10	12	13	10	37	12	12	10	11	13	298	146	126	333	306	155	155	346	346	154	133	345	320	160	160	358
1011100	349	370	208	221	418	380	241	191	480	350	257	212	473	410	207	209	462	413	202	232	505	380	232	242	495	419	253	211	500	445	246	214	485
1011100	350	13	14	33	15	12	14	13	15	13	12	14	15	13	15	34	19	12	12	21	20	12	12	13	21	13	13	14	16	14	13	14	49
1011100	351	12	13	12	15	12	22	14	15	20	13	14	15	12	13	19	15	13	13	13	18	12	12	13	22	14	12	14	16	13	14	14	16
1011101	352	2	1	2	2	2	0	2	2	3	0	2	2	2	1	2	2	2	0	2	2	2	1	2	2	2	0	2	2	2	0	2	2
1011101	353	75	112	77	105	77	75	77	83	83	81	83	84	84	91	85	86	86	84	86	89	88	96	90	104	96	98	90	94	96	91	92	95
1011110	354	1610	1594	1672	1629	1672	1678	1747	1808	1781	1769	1871	1828	1836	1843	1862	1878	1901	1885	1931	1947	1934	1994	2083	2027	2023	2081	2157	2052	2065	2099	2081	2059
1011110	355	2	2	2	3	2	2	2	3	2	2	2	3	2	2	1	3	2	2	2	3	2	2	2	3	2	4	2	3	2	2	2	3
1011111	356	975	931	2	974	1010	1018	2	1055	1071	1074	2	1091	1074	1036	2	1104	1083	1064	2	1103	1132	1142	2	1182	1188	1148	1	1160	1199	1168	3	1242
1011111	357	10	8	10	11	12	9	10	12	11	11	10	13	11	9	10	14	12	9	11	12	12	9	11	13	14	11	12	13	13	10	12	13
1011111	358	3392	3377	3480	3479	3460	3507	3520	3736	3700	3723	3756	3806	3873	3857	3869	3922	3947	3938	3992	4023	4025	4012	4211	4252	4280	4405	4154	4215	4284	4370	4302	4321
1100000	359	289	99	2	317	269	103	4	316	319	109	1	343	300	137	1	355	324	112	2	331	303	136	1	367	329	125	2	344	326	122	2	383
1100000	360	22	15	2	74	29	22	1	79	23	16	2	45	30	22	1	52	23	18	2	95	61	25	2	79	25	19	1	46	32	23	1	84
1100000	361	308	114	1	300	310	109	2	316	300	116	2	329	338	119	2	353	337	129	2	340	339	123	2	357	360	132	1	355	360	130	1	358
1100001	362	327	139	2	385	341	146	2	410	364	120	2	381	369	154	2	404	528	391	2	607	573	368	2	681	604	385	2	634	604	425	2	712
1100001	363	272	134	137	368	279	113	140	353	298	145	122	357	307	121	133	340	330	116	121	369	310	122	152	353	356	127	130	413	356	146	156	472
1100001	364	44	16	2	35	31	21	2	72	23	19	2	32	30	21	2	42	44	21	2	49	97	25	2	56	41	22	2	51	44	50	2	62
1100010	365	301	129	130	343	333	132	163	356	328	139	172	368	333	176	144	394	360	140	181	399	312	143	150	404	394	177	181	421	357	174	184	395
1100010	366	308	101	133	382	315	104	106	357	327	111	127	365	342	112	115	402	307	112	134	439	337	114	119	412	350	119	122	444	340	137	123	429
1100010	367	272	98	2	370	287	101	2	410	299	109	2	364	307	111	2	394	331	109	2	378	359	128	2	478	378	115	2	390	357	147	2	397
1100011	368	325	123	2	302	307	152	2	325	300	200	1	346	330	140	1	329	338	142	1	339	314	179	1	407	336	155	2	395	360	178	2	384
1100011	369	304	158	158	391	321	166	142	377	360	142	146	397	405	151	151	413	351	209	172	449	344	147	159	359	372	225	188	450	370	178	183	433
1100011	370	13	10	2	15	13	19	2	15	14	11	2	16	15	11	2	16	14	11	2	16	15	11	2	17	14	14	2	18	15	13	2	18
1100100	371	283	132	136	362	314	135	146	406	310	144	146	385	318	148	148	415	331	170	146	392	349	172	153	412	385	175	164	406	330	209	166	386
1100100	372	337	139	124	431	288	150	157	435	363	156	159	392	345	135	136	428	368	132	163	457	319	134	142	452	359	148	172	449	367	142	144	447
1100100	373	282	182	160	351	297	130	131	419	334	135	137	419	342	196	168	401	322	165	138	388	373	138	140	410	384	149	169	440	342	146	172	429

continued on next page

Table A.4: *CitiesAuto* absolute (ms): average over all iterations of a query without DB intervals
(continued from previous page)

class	ID	−DB intervals																															
		+indirect keyword occurrences																−indirect keyword occurrences															
		+index intervals								−index intervals								+index intervals								−index intervals							
		+label index				−label index				+label index				−label index				+label index				−label index				+label index				−label index			
		ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG
		32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
1100101	374	34	25	26	47	46	30	32	51	37	27	29	43	47	32	34	81	38	28	32	43	48	33	35	53	40	29	33	45	74	33	37	56
1100101	375	298	127	134	342	304	131	158	330	347	139	142	360	317	143	144	341	329	164	141	373	309	168	144	419	322	148	150	359	329	149	151	387
1100101	376	323	124	1	381	302	128	1	414	330	158	2	380	348	140	1	383	321	137	3	387	325	140	2	389	406	145	2	413	374	148	2	416
1100110	377	272	155	130	341	334	131	133	383	325	141	169	377	332	170	173	392	317	143	142	407	364	144	147	416	394	180	180	414	385	151	155	451
1100110	378	293	160	132	351	280	131	163	413	330	140	143	417	310	174	151	378	334	143	198	375	312	164	155	446	329	166	153	415	342	193	161	394
1100110	379	301	130	161	369	306	132	159	368	328	140	145	366	343	145	145	369	342	150	146	374	390	152	162	419	354	208	151	397	357	164	156	448
1100111	380	31	13	14	30	34	21	23	39	31	14	15	58	38	21	21	37	32	13	16	34	44	20	23	42	36	20	18	34	68	22	22	41
1100111	381	22	9	2	26	33	12	1	35	25	9	3	27	32	14	2	36	52	10	2	29	35	13	2	35	29	9	1	34	34	13	2	35
1100111	382	560	234	2	649	573	217	2	672	619	235	2	680	660	239	2	723	647	242	2	713	649	247	2	730	693	281	2	736	690	348	2	846
1101000	383	4	4	2	5	9	12	1	11	4	4	2	5	11	11	2	11	56	9	2	56	88	13	2	94	66	8	2	58	102	19	1	63
1101000	384	16	15	1	24	22	23	2	31	18	16	2	24	23	22	1	31	29	20	1	36	35	76	2	47	46	21	2	37	38	31	2	46
1101000	385	22	14	2	43	29	21	2	50	23	17	2	44	29	52	1	51	23	17	1	46	32	29	1	87	27	18	1	48	30	24	1	55
1101001	386	22	14	16	25	31	18	19	64	24	15	16	27	36	19	21	35	28	20	22	30	35	23	26	41	29	20	22	31	37	24	25	38
1101001	387	25	3	5	54	31	11	12	38	29	4	5	31	36	11	11	39	29	5	5	30	38	12	13	41	30	5	6	33	40	11	13	40
1101001	388	319	114	2	334	327	119	1	386	320	126	1	386	366	130	2	418	357	126	2	402	370	128	2	410	354	159	2	422	387	149	2	442
1101010	389	29	14	2	36	38	16	2	43	34	16	2	36	40	39	2	44	155	95	2	135	137	99	2	173	151	100	3	170	143	119	2	246
1101010	390	2	3	2	4	8	9	2	10	2	2	2	2	7	13	2	9	2	2	1	3	8	8	2	10	2	2	2	3	8	8	2	8
1101010	391	315	140	143	494	334	150	189	508	383	211	180	509	326	159	186	498	1914	1729	1793	2030	1988	1839	1909	2179	2121	1913	1881	2131	2075	1939	1915	2285
1101011	392	27	18	18	19	25	25	25	28	19	19	19	20	26	54	26	56	21	19	20	22	28	26	29	31	22	20	22	54	31	26	28	36
1101011	393	11	10	12	12	18	17	18	19	11	12	14	13	17	17	17	18	295	169	137	357	314	143	159	405	316	185	144	360	358	151	153	418
1101011	394	21	12	14	26	22	14	14	26	24	14	15	27	26	14	15	26	27	15	17	29	28	16	19	30	47	19	17	34	28	17	20	40
1101100	395	7383	7454	7539	7701	7753	7733	7788	8358	8387	8386	8420	8571	8646	8645	8666	8868	8804	8845	8897	9049	9133	9166	9572	9660	9713	9742	9564	9618	9801	9742	9739	10614
1101100	396	12	13	44	12	16	17	49	18	15	46	15	16	19	19	18	18	15	16	15	14	19	17	18	19	18	18	17	13	20	19	19	19
1101100	397	1	0	0	0	10	9	10	10	1	1	1	0	9	9	9	9	0	1	9	0	10	10	10	10	0	0	0	1	9	9	10	9
1101101	398	3	2	3	2	8	8	11	9	2	2	3	2	7	13	10	8	2	2	3	2	8	8	10	9	2	2	3	2	8	8	9	8
1101101	399	2	2	3	2	2	2	3	3	2	2	4	2	3	2	4	2	2	2	3	2	2	2	4	2	2	2	4	2	3	2	3	2
1101110	400	2	2	3	4	9	6	8	8	2	2	4	2	8	6	7	9	2	2	3	4	9	5	7	10	2	2	3	3	8	7	8	11
1101110	401	6	1	2	8	11	5	7	14	7	1	2	9	12	5	9	13	7	1	3	10	12	6	7	69	7	0	2	10	12	5	6	14

continued on next page

Table A.4: *CitiesAuto* absolute (ms): average over all iterations of a query without DB intervals
(continued from previous page)

class	ID	−DB intervals																															
		+indirect keyword occurrences																−indirect keyword occurrences															
		+index intervals								−index intervals								+index intervals								−index intervals							
		+label index				−label index				+label index				−label index				+label index				−label index				+label index				−label index			
		ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG
		32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
1101110	402	17	0	22	26	52	6	7	32	19	1	2	27	25	6	6	29	19	1	2	27	25	5	7	33	21	1	2	29	25	4	35	35
1101111	403	283	102	1	323	271	104	1	339	320	112	1	319	301	118	2	353	325	116	2	357	303	141	2	350	388	124	1	347	358	128	2	375
1101111	404	45	4	1	16	19	6	2	22	17	4	2	19	19	7	2	21	17	3	2	19	23	6	2	23	19	4	1	19	23	6	2	23
1101111	405	2	1	3	2	9	8	8	8	2	0	1	2	8	6	9	9	2	0	3	2	8	8	8	9	2	1	2	2	9	6	8	8
1110000	406	12	10	10	16	11	9	11	15	12	10	12	16	13	10	13	17	12	10	11	18	38	9	12	16	12	10	11	17	14	10	12	16
1110000	407	52	48	50	65	59	59	52	68	57	54	55	69	59	55	81	71	67	88	59	70	68	61	62	99	71	87	69	76	82	69	65	77
1110000	408	1180	1148	1198	1262	1218	1202	1274	1320	1312	1299	1319	1365	1368	1376	1377	1389	3730	3551	3580	3892	3854	3710	3885	4154	4123	3772	3885	4155	4236	3935	3879	4276
1110001	409	462	224	2	543	478	210	2	576	538	258	1	576	518	224	2	594	665	410	3	730	676	420	2	760	738	407	2	761	765	479	2	797
1110001	410	6	7	2	7	16	12	2	18	7	7	2	8	16	10	2	15	6	7	2	8	16	10	2	17	7	7	2	8	19	10	2	17
1110001	411	87	62	63	144	101	93	71	125	93	67	99	145	98	67	70	118	509	285	314	599	518	290	307	609	546	299	329	608	585	362	351	644
1110010	412	6095	6129	6263	6333	6310	6322	6421	6801	6747	6806	6884	7000	6990	6932	7046	7181	7361	7244	7274	7539	7556	7432	7758	8024	8011	7528	8487	8003	8669	7862	7859	8210
1110010	413	7294	7376	7491	7654	7603	7608	7711	8186	8240	8267	8290	8479	8455	8452	8525	8716	8727	8691	8784	8929	8924	8943	9364	9525	9512	9532	9662	9509	9910	9504	9528	9749
1110010	414	7322	7370	7497	7647	7615	7625	7707	8273	8145	8236	8275	8432	8494	8487	8524	8700	8661	8718	8787	8969	8928	8930	9335	9451	9462	9553	10199	9499	9649	9457	9519	10317
1110011	415	60	49	53	84	61	54	54	68	65	80	56	100	69	95	58	74	402	204	208	448	385	234	242	477	438	276	263	441	449	255	224	472
1110011	416	11	9	10	17	11	11	12	12	11	9	11	12	12	10	11	13	12	10	12	14	13	9	11	13	12	11	11	15	13	10	11	13
1110011	417	52	4	6	29	27	7	6	30	56	5	6	31	30	4	7	32	63	5	7	54	55	6	8	76	58	5	7	58	58	5	7	57
1110100	418	13	9	12	15	12	11	11	16	12	10	12	17	12	10	11	18	11	10	11	16	13	10	11	17	13	9	13	16	18	10	11	18
1110100	419	11	9	12	14	12	11	11	46	12	10	11	17	12	10	11	18	12	10	11	18	12	9	11	17	12	10	14	16	12	10	12	57
1110100	420	11	9	10	14	11	11	16	14	12	10	11	14	12	10	11	45	11	13	10	14	11	11	11	15	12	11	14	37	12	10	12	15
1110101	421	11	9	15	11	12	11	10	32	12	10	11	13	12	11	12	12	12	9	12	14	12	9	10	13	13	10	11	18	14	41	11	13
1110101	422	26	4	5	28	26	5	5	28	28	5	5	29	30	4	6	30	31	4	5	30	49	5	7	31	30	5	6	32	33	4	8	32
1110101	423	11	9	10	12	13	9	11	12	12	11	11	13	12	10	13	13	12	9	12	13	12	10	11	13	12	10	11	14	12	10	37	14
1110110	424	262	117	121	332	294	121	154	348	289	161	158	376	329	133	134	406	294	157	134	390	331	134	145	396	324	138	152	377	357	144	142	402
1110110	425	72	54	54	102	74	54	60	159	77	62	59	114	81	62	85	107	111	93	65	146	90	62	68	151	83	67	97	169	86	67	66	129
1110110	426	108	57	54	107	69	59	61	106	99	72	61	106	80	59	65	107	80	60	64	109	107	59	72	126	83	82	69	111	91	95	64	193
1110111	427	17	11	12	32	14	12	12	28	14	12	13	29	16	12	13	29	15	11	19	29	16	12	41	40	16	12	14	28	18	13	14	30
1110111	428	19	12	13	21	16	11	13	23	18	13	15	23	18	13	15	23	290	124	114	327	286	182	133	390	338	125	179	400	322	123	127	420
1110111	429	18	10	11	20	26	10	11	23	50	12	12	21	19	11	13	22	21	12	12	53	20	11	13	26	22	11	14	24	23	12	14	28

continued on next page

Table A.4: *CitiesAuto* absolute (ms): average over all iterations of a query without DB intervals
(continued from previous page)

class	ID	−DB intervals															
		+indirect keyword occurrences															
		+index intervals								−index intervals							
		+label index				−label index				+label index				−label index			
		ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG
		32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
1111000	430	64	60	62	120	117	62	64	125	70	91	68	96	97	67	69	128
1111000	431	11	9	41	18	17	10	10	15	13	10	11	16	13	11	12	15
1111000	432	1	0	1	1	1	1	1	1	1	1	0	1	0	1	1	0
1111001	433	15	12	23	26	14	12	13	29	15	14	15	59	15	13	15	37
1111001	434	1	0	2	0	0	1	2	1	0	0	2	0	1	0	2	1
1111001	435	13	12	33	13	12	11	12	15	13	13	14	14	13	12	15	14
1111010	436	24	29	16	52	15	10	12	27	17	10	12	27	18	11	13	27
1111010	437	457	231	230	669	416	262	271	686	468	281	278	697	475	283	309	695
1111010	438	13	8	9	24	17	8	10	21	14	10	9	20	14	8	10	25
1111011	439	115	116	146	120	115	112	130	125	128	145	121	153	130	127	138	160
1111011	440	16	15	15	24	17	14	15	27	48	16	16	27	18	16	18	23
1111011	441	12	9	11	13	12	10	12	14	13	11	12	13	14	10	12	13
1111100	442	1	1	1	0	0	0	1	0	0	0	0	0	1	1	1	0
1111100	443	0	1	1	1	0	0	1	1	1	0	0	0	1	1	1	1
1111100	444	2	3	3	2	2	2	2	3	2	2	2	3	2	2	3	3
1111101	445	1	2	3	2	2	2	3	2	2	2	3	2	2	2	3	2
1111101	446	2	2	3	2	2	3	3	2	2	2	3	2	2	2	4	2
1111110	447	3	1	2	4	3	1	2	5	3	1	2	5	3	1	2	5
1111110	448	2	2	4	3	2	2	4	33	2	2	4	3	2	2	3	3
1111111	449	2	1	2	2	2	0	1	2	2	1	2	2	2	0	2	2
1111111	450	2	0	2	3	3	0	1	2	2	0	2	2	2	1	1	2
1111111	451	0	1	1	1	1	0	2	1	0	0	1	0	1	1	1	1

class	ID	−DB intervals															
		−indirect keyword occurrences															
		+index intervals								−index intervals							
		+label index				−label index				+label index				−label index			
		ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG
		48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
1111000	430	89	79	80	98	90	80	83	107	100	82	95	102	106	86	87	109
1111000	431	12	9	11	17	13	11	11	17	12	10	13	17	14	10	12	16
1111000	432	1	0	1	0	0	1	0	1	0	1	1	0	0	2	1	0
1111001	433	22	16	18	43	23	16	17	44	27	16	19	45	27	16	18	47
1111001	434	0	1	2	1	1	0	2	0	0	1	2	0	1	0	2	0
1111001	435	439	246	275	447	466	251	318	493	440	284	268	521	442	291	270	557
1111010	436	22	13	23	36	23	14	14	28	31	15	16	30	26	14	15	30
1111010	437	3429	3119	3179	3598	3507	3208	3355	3893	3764	3332	3407	3833	4162	3391	3414	4090
1111010	438	21	11	14	27	22	12	15	28	29	11	19	27	25	13	13	28
1111011	439	137	138	131	179	145	131	150	148	144	163	163	181	176	146	145	163
1111011	440	29	19	21	38	30	21	22	43	31	27	25	38	31	22	23	45
1111011	441	360	195	196	379	388	172	210	416	393	193	186	458	375	186	185	407
1111100	442	0	1	1	0	0	1	0	0	0	0	1	1	1	0	0	2
1111100	443	1	1	1	0	0	0	1	0	1	0	1	0	2	0	1	1
1111100	444	2	3	3	2	2	3	4	2	2	2	3	2	3	4	2	3
1111101	445	2	2	3	3	2	2	3	2	2	3	33	2	2	2	4	2
1111101	446	2	2	3	3	2	2	3	2	2	2	4	2	2	2	4	2
1111110	447	3	1	1	5	3	0	2	5	3	1	3	6	3	0	2	5
1111110	448	2	3	4	4	2	2	3	3	2	2	3	3	2	32	3	3
1111111	449	2	1	2	2	2	0	2	2	2	1	2	2	2	0	2	2
1111111	450	2	0	2	2	2	0	2	2	2	0	2	2	2	0	2	2
1111111	451	0	1	2	0	1	1	2	0	0	1	1	1	1	0	2	1

Table A.5: *XMarkAuto* absolute (ms): average over all iterations of a query with DB intervals

class	ID	+DB intervals																															
		+indirect keyword occurrences																−indirect keyword occurrences															
		+index intervals								−index intervals								+index intervals								−index intervals							
		+label index				−label index				+label index				−label index				+label index				−label index				+label index				−label index			
		ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG
		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0000011	800	1358	1307	1050	24234	1344	1121	1140	23110	1459	1217	1205	23589	1494	1257	1291	23491	1563	1310	1324	23340	1631	1350	1353	23571	1692	1420	1464	23626	1779	1472	1473	23596
0010100	801	4450	4479	4686	4644	4829	4767	4812	4918	4849	4958	4948	5042	4968	5070	5075	5209	5229	5354	5448	5443	5393	5578	5526	5683	5655	5721	5695	5744	5835	5846	5902	5914
0010100	802	32	30	64	30	33	31	29	28	30	37	34	31	31	33	33	32	31	43	33	60	33	36	68	36	35	40	50	32	36	37	37	35
0010110	803	681	523	596	8006	848	553	526	7966	702	578	564	7938	716	638	657	7951	680	639	608	8027	748	655	663	7961	730	659	634	8014	780	619	655	8019
0011011	804	473	387	374	7888	546	354	368	7786	491	435	394	7868	528	391	395	7857	602	479	491	7935	587	477	474	7859	618	516	531	7793	629	534	541	7863
0011100	805	347	369	346	418	474	359	387	390	391	404	373	381	408	409	455	421	428	463	411	471	472	452	451	455	431	498	464	466	474	476	476	482
0011100	806	25	17	18	17	18	19	19	18	19	19	19	20	20	19	22	56	23	22	22	22	23	23	23	22	21	21	22	21	22	22	22	21
0011100	807	8	7	6	7	8	7	7	7	7	7	7	7	7	7	7	9	7	7	8	7	7	8	7	7	8	8	8	7	7	8	8	7
0011100	808	330	380	404	456	483	382	388	373	401	395	380	407	409	421	420	424	406	435	444	451	454	433	462	459	440	466	468	512	487	476	491	519
0011100	809	7	7	16	5	8	7	7	6	7	6	7	6	7	9	9	6	72	10	8	6	10	7	7	6	7	8	7	8	7	8	8	8
0011110	810	3333	3437	3478	3593	3953	3566	3615	3773	3710	3770	3704	3885	3833	3883	3895	4028	4060	3985	4109	4236	4175	4148	4261	4353	4283	4298	4348	4492	4427	4438	4482	4605
0011111	811	1233	1238	1317	1401	1497	1291	1315	1470	1358	1369	1369	1471	1411	1429	1450	1538	1476	1448	1502	1582	1534	1545	1562	1705	1527	1544	1562	1710	1632	1614	1620	1796
0011111	812	553	625	530	675	670	606	590	736	611	568	621	708	646	622	631	738	639	646	648	763	673	709	681	789	714	663	726	794	708	756	746	827
0100011	813	435	404	361	8047	587	376	377	8258	556	397	426	8155	558	439	416	8220	1385	1255	1268	8744	1410	1295	1307	8872	1435	1341	1366	9034	1504	1397	1404	9130
0100100	814	17003	18124	18253	18686	18253	18285	18458	18688	18913	18952	19233	19313	19686	19962	20179	20105	20755	21028	21069	21131	21501	21794	21915	21968	22307	22402	22614	22686	23185	23308	23433	23450
0100101	815	1337	1380	1325	8972	1426	1358	1406	8052	1522	1421	1470	8760	1550	1469	1493	8832	1585	1523	1535	8877	1640	1573	1599	8450	1737	1637	1653	9147	1785	1701	1730	8859
0100101	816	4255	4370	4439	12144	4600	4492	4607	12166	4731	4744	4807	12216	4989	4935	4976	12638	5183	5209	5307	12741	5465	5378	5489	13128	5672	5575	5679	13263	5926	5886	5872	13441
0100101	817	469	388	422	8152	458	384	391	7763	508	433	413	7792	511	420	461	7854	490	464	412	7917	553	429	452	8081	567	469	457	8034	580	493	461	7940
0100110	818	507	425	481	7882	469	412	432	7469	536	454	434	7691	539	494	475	7787	590	464	456	7758	563	458	486	7728	623	558	494	7838	623	529	571	7895
0100111	819	415	359	347	7683	417	375	380	7672	489	402	385	7750	503	392	428	7937	492	400	399	7829	509	405	439	7762	526	467	450	8020	566	484	445	7784
0100111	820	431	338	365	8453	461	374	411	7775	497	378	411	7750	501	415	423	7717	515	393	424	7737	521	419	411	7886	525	458	464	8062	553	475	459	7786
0100111	821	833	728	746	16212	891	743	759	15562	992	800	838	15806	1026	853	850	15908	1047	841	876	15785	1045	857	874	15664	1130	944	957	16069	1173	979	947	16139
0100111	822	468	396	399	8007	482	385	415	7690	518	446	441	7976	533	460	460	7957	542	457	467	7898	561	471	476	7757	615	521	502	7861	612	515	488	8030
0101001	823	82	95	117	198	104	102	106	172	93	89	95	178	120	106	117	169	116	117	117	164	161	135	136	207	124	125	126	222	145	145	151	214
0101011	824	80	61	59	288	76	75	102	318	67	63	64	294	87	75	75	308	74	69	72	304	96	84	117	323	83	75	80	308	101	87	91	335
0101100	825	4457	4764	4646	5182	4808	4804	4868	5019	4940	5010	5015	5086	5138	5171	5272	5373	5408	5514	5521	5677	5544	5699	5635	5821	5765	5797	5881	5985	6027	6093	6096	6222
0101100	826	7895	8170	8248	8432	8264	8415	8480	8673	8584	8622	8834	8920	8934	9002	9075	9361	9421	9581	9626	9762	9929	9776	9872	10043	10088	10124	10209	10406	10423	10565	10486	10649
0101100	827	4930	4620	4483	5021	4567	4602	4755	4749	4732	4895	4893	4890	4968	4994	5039	5098	5224	5445	5347	5371	5441	5523	5473	5508	5570	5622	5680	5704	5738	5847	5805	5904

continued on next page

Table A.5: *XMarkAuto* absolute (ms): average over all iterations of a query with DB intervals
(continued from previous page)

class	ID	+DB intervals																															
		+indirect keyword occurrences																−indirect keyword occurrences															
		+index intervals								−index intervals								+index intervals								−index intervals							
		+label index				−label index				+label index				−label index				+label index				−label index				+label index				−label index			
		ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG
		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0101101	828	3	3	4	7	22	19	17	25	3	3	4	13	13	38	25	17	3	3	4	10	17	21	15	18	4	4	5	11	23	14	15	21
0101110	829	4580	4591	4608	5174	4736	4822	4872	4935	4956	5023	5039	5041	5183	5085	5253	5330	5358	5532	5504	5504	5634	5678	5713	5740	5731	5883	5819	5940	6006	6119	5996	6203
0101110	830	25	20	24	70	64	36	40	124	21	49	25	60	34	32	38	76	49	24	27	95	60	49	43	78	32	26	31	113	42	37	47	82
0101111	831	4	3	4	4	15	16	17	20	3	3	4	3	15	39	18	15	4	3	5	3	16	14	21	19	3	3	5	4	15	26	15	16
0101111	832	3	3	4	4	18	37	19	20	3	3	4	4	15	15	16	16	3	3	4	4	19	18	21	18	3	3	5	4	15	17	16	16
0101111	833	3	3	4	4	15	16	21	14	3	3	4	3	15	13	16	15	4	3	4	4	19	19	51	15	4	5	39	4	15	15	15	18
0101111	834	4	4	5	6	17	17	20	22	4	4	6	5	17	16	23	18	5	5	6	6	16	53	21	63	5	5	6	7	17	15	18	62
0101111	835	8	7	8	8	21	20	25	25	7	7	9	8	19	19	20	23	8	8	9	9	64	49	31	23	8	8	9	9	21	22	67	21
0101111	836	10	11	12	13	27	27	29	28	12	12	14	13	24	24	28	26	16	15	14	14	34	30	31	30	14	39	37	15	27	28	28	28
0101111	837	3	3	5	3	17	16	19	19	3	3	6	3	13	17	16	16	3	5	4	4	16	17	51	20	3	4	5	4	15	14	17	17
0110001	838	3038	3148	3102	11674	3304	3237	3244	11058	3427	3411	3438	11222	3589	3447	3541	11352	5946	5796	5976	19056	6145	6058	6215	19331	6390	6270	6349	19516	6704	6534	6505	19632
0110001	839	124	135	154	690	167	136	161	614	165	138	141	618	145	139	203	668	320	345	349	1428	388	346	362	1466	337	342	367	1457	389	390	357	1455
0110011	840	50	33	33	2334	44	33	35	2298	47	36	41	2346	54	39	43	2275	551	453	425	8851	529	419	432	8909	525	423	461	8971	547	465	463	8775
0110101	841	64	64	72	109	74	104	74	144	77	77	78	167	73	78	80	118	77	78	80	124	80	80	89	163	111	116	87	161	107	88	88	133
0110110	842	7623	7953	8157	10406	8282	8322	8314	10173	8544	8550	8717	10420	8865	8778	8977	10675	9255	9368	9495	11123	9612	9605	9765	11510	9836	9939	9967	11650	10287	10386	10273	11984
0110111	843	3	2	4	40	3	3	4	45	3	3	4	43	3	3	4	77	3	3	4	42	3	4	4	42	4	4	4	86	4	4	5	46
0110111	844	5	5	4	41	5	5	4	81	3	3	4	45	3	23	6	43	3	3	4	43	3	5	4	41	3	3	6	41	4	3	5	79
0110111	845	3	6	4	42	3	3	4	44	3	3	4	81	3	3	4	68	3	3	4	43	3	3	4	41	3	3	4	42	4	3	5	44
0110111	846	549	516	533	2015	529	490	559	1944	562	529	569	2009	440	490	541	1941	532	466	458	1952	441	465	542	1995	578	479	515	2110	494	533	470	2071
0111000	847	4939	5041	5279	5741	5359	5385	5444	5942	5534	5608	5630	6262	5783	5814	5878	6397	6396	6412	6521	7145	6630	6664	6724	7343	6802	6870	6915	7450	7170	7200	7173	7766
0111001	848	9198	9228	9499	10978	9598	9558	9730	11216	9903	10016	10107	11574	10394	10373	10521	11960	10860	10994	11112	12486	11291	11284	11452	13005	11577	11648	11746	13182	12082	12139	12178	13493
0111001	849	11	11	15	33	11	11	13	34	12	12	13	35	12	20	13	33	18	15	17	30	23	15	17	31	23	20	21	27	21	19	17	69
0111001	850	288	251	297	275	270	324	300	296	305	316	319	324	350	297	331	335	368	351	394	408	371	336	372	445	391	412	360	417	407	527	364	430
0111100	851	11	8	10	8	9	8	9	8	9	9	46	8	11	9	11	9	9	9	9	9	9	32	10	10	9	12	30	19	48	10	11	13
0111100	852	334	314	334	332	370	338	374	344	386	360	381	381	389	389	377	365	386	386	416	422	427	410	471	416	438	482	416	420	457	558	434	498
0111110	853	2221	2243	2359	2354	2426	2345	2405	2507	2496	2503	2501	2562	2536	2583	2610	2644	2660	2705	2758	2727	2790	2774	2767	2818	2856	2898	2903	2961	2951	3019	2960	3035
1000001	854	803	673	684	10373	847	699	719	10230	923	752	751	10375	904	758	808	10409	965	822	810	10334	965	827	825	10429	1068	900	866	10609	1055	894	895	10412
1000001	855	625	525	536	9047	674	529	557	8871	695	555	569	8931	712	603	607	8910	744	604	574	9046	717	596	647	8899	780	626	624	9009	785	645	648	8916

continued on next page

Table A.5: *XMarkAuto* absolute (ms): average over all iterations of a query with DB intervals
(continued from previous page)

class	ID	+DB intervals																															
		+indirect keyword occurrences																−indirect keyword occurrences															
		+index intervals								−index intervals								+index intervals								−index intervals							
		+label index				−label index				+label index				−label index				+label index				−label index				+label index				−label index			
		ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG
		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
1000010	856	454	340	348	7916	574	333	343	7962	589	362	366	7938	590	395	413	7984	593	398	417	8017	615	419	390	7992	641	410	449	8009	666	434	433	8047
1000011	857	425	365	354	7770	452	365	331	7971	472	357	384	7908	499	386	394	7861	516	370	395	8012	517	399	409	7788	550	429	412	7958	569	447	415	7864
1000011	858	884	668	721	15355	967	684	719	15515	994	731	775	15739	1057	754	769	15489	1703	1655	1618	9226	1739	1679	1698	9436	1833	1751	1758	9296	1900	1812	1847	9532
1000011	859	476	365	371	8996	500	377	392	8997	541	390	385	8948	583	394	399	8977	657	508	521	9013	643	536	534	9007	666	587	573	8932	699	574	577	9113
1000101	860	726	570	617	9694	770	629	634	9467	836	665	670	9634	802	700	697	9607	812	677	684	9686	854	681	697	9654	888	729	759	9669	886	783	752	9705
1000101	861	598	511	526	8749	654	509	514	8916	680	596	575	8739	738	628	622	9011	698	571	603	8771	709	583	564	8836	752	614	654	8878	781	677	638	8868
1000101	862	471	379	401	7863	460	369	381	7735	523	426	425	7789	528	424	433	7798	521	420	434	7896	574	404	416	8083	593	488	459	8041	617	472	478	7936
1000111	863	426	326	311	7659	424	359	327	7813	487	374	379	7775	514	360	421	7788	510	400	423	7864	517	380	372	7890	519	394	406	7871	566	435	460	7988
1000111	864	401	303	338	7938	429	345	354	7969	494	379	378	8067	476	354	396	7995	536	385	365	7986	526	396	398	7887	524	422	398	7899	572	462	435	8017
1000111	865	449	296	359	7971	465	310	380	7881	516	348	398	7968	481	375	356	7970	515	350	365	7966	521	387	390	7916	583	425	421	8070	556	424	465	8092
1001100	866	6	41	12	6	19	50	19	19	5	8	5	8	46	18	48	16	6	6	6	6	18	19	19	19	13	6	7	8	17	17	16	17
1001100	867	6	4	4	5	15	38	13	13	5	5	5	3	14	11	14	47	35	5	3	5	14	14	11	13	5	3	3	5	14	43	11	12
1001101	868	1	3	2	1	17	18	17	16	1	1	2	1	16	17	17	40	1	3	3	1	17	16	17	17	1	4	3	1	65	15	17	17
1001101	869	7981	8086	1	9336	8516	8555	1	9782	8734	8866	3	10022	9121	9201	1	10346	9519	9677	1	10789	9878	9989	1	11069	10214	10275	1	11398	10607	10669	1	11728
1001110	870	477	313	2	7681	543	356	2	7707	600	386	2	7743	622	370	2	7935	577	397	1	7854	566	421	2	7675	597	413	2	7960	621	460	3	7835
1001111	871	426	351	1	7825	413	343	2	7648	449	393	1	7836	490	410	2	7761	510	386	3	7670	524	400	1	7770	578	435	1	7840	595	463	1	7884
1010001	872	619	514	514	8886	641	540	553	8934	693	548	577	8853	709	600	603	9070	811	660	707	8827	822	690	732	9152	872	715	728	9228	902	735	754	9382
1010011	873	517	407	390	11331	560	419	460	11354	592	436	468	11353	609	465	473	11358	1565	1275	1331	23628	1604	1372	1400	23998	1728	1421	1445	23785	1798	1463	1510	23673
1010011	874	553	397	387	10163	533	418	428	10183	576	461	490	10196	573	508	509	10089	1574	1344	1338	23246	1614	1353	1376	23026	1733	1432	1471	23506	1779	1470	1490	23316
1010011	875	401	297	344	7946	448	313	329	7707	485	343	364	7814	469	407	361	7831	485	375	387	7796	499	390	428	7943	549	414	423	8011	559	405	408	7818
1010101	876	582	468	530	8847	604	515	514	8699	652	509	576	8927	668	561	598	8907	659	535	550	8991	668	558	569	8891	710	587	622	9025	729	596	606	9048
1010101	877	669	491	594	9085	652	534	542	9019	715	537	574	8912	701	580	589	8934	696	565	585	8977	715	585	592	8853	746	615	658	8963	778	637	644	8968
1010110	878	8	4	2	7	6	5	9	7	6	6	1	9	6	6	2	40	6	5	2	8	6	6	3	8	7	6	2	8	7	6	33	10
1010110	879	6	5	2	6	6	5	2	7	6	5	2	9	6	6	3	9	6	5	2	8	6	5	1	8	7	6	4	8	7	7	1	9
1010111	880	424	319	2	7599	446	341	2	7667	488	343	2	7649	502	378	2	7883	532	385	2	7700	510	363	2	7835	552	408	2	7858	609	423	2	7819
1010111	881	439	333	339	7768	449	332	325	7773	476	370	375	7782	496	362	387	7699	500	356	394	7835	556	396	401	7751	581	403	423	7803	567	429	433	7791
1010111	882	486	304	2	7659	468	381	2	7499	478	343	2	7807	517	366	2	7693	521	423	2	7678	536	488	2	7618	574	451	2	7687	625	463	2	7692
1011000	883	523	359	1	7516	461	356	1	7486	470	394	1	7662	507	422	1	7684	530	412	1	7735	518	452	1	7676	568	443	2	7702	587	452	2	7846

continued on next page

Table A.5: *XMarkAuto* absolute (ms): average over all iterations of a query with DB intervals
(continued from previous page)

class	ID	+DB intervals																															
		+indirect keyword occurrences																−indirect keyword occurrences															
		+index intervals								−index intervals								+index intervals								−index intervals							
		+label index				−label index				+label index				−label index				+label index				−label index				+label index				−label index			
		ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG
		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
1011000	884	3595	3359	2	3544	3594	3619	1	3706	3717	3725	1	3892	3811	3829	2	4094	4628	4450	1	11510	4704	4677	2	11410	4882	4858	1	11754	5172	4931	1	11987
1011001	885	7	5	2	6	26	5	2	7	6	5	1	7	7	5	1	7	520	394	1	7747	515	404	1	7688	512	433	1	7851	598	468	2	7840
1011001	886	498	487	511	8007	547	467	455	8126	577	501	499	8090	595	548	512	8123	2496	2470	2491	9907	2606	2533	2584	9842	2671	2609	2639	10085	2817	2717	2726	10179
1011001	887	692	706	825	804	748	737	754	870	775	777	788	860	808	807	848	900	868	853	922	935	957	912	951	958	959	922	922	1049	970	976	994	1014
1011010	888	5	5	1	7	6	5	1	7	7	5	1	9	7	6	2	9	490	413	2	7776	506	445	1	7869	546	449	1	7805	589	463	1	7888
1011010	889	6	5	1	6	6	5	2	7	6	5	1	7	8	6	1	7	492	377	1	7596	508	391	1	7728	541	440	2	8164	569	428	2	7665
1011011	890	275	216	274	2758	317	228	259	2627	316	245	253	2614	340	257	262	2651	362	282	262	2638	392	291	303	2702	421	283	311	2715	369	288	319	2683
1011011	891	412	347	4	7605	501	378	2	7642	504	368	2	7765	514	385	1	7940	524	459	1	7667	537	451	1	7685	567	492	2	7745	585	508	1	7662
1011011	892	1254	1153	1266	8304	1348	1278	1277	8288	1388	1265	1354	8513	1455	1371	1416	8364	1545	1406	1423	8521	1621	1491	1490	8480	1634	1511	1579	8743	1756	1567	1577	8713
1011100	893	7	6	7	665	23	21	24	681	8	6	10	631	51	20	22	643	10	6	8	648	24	23	25	672	9	10	8	652	24	21	22	653
1011100	894	19	50	18	198	30	29	31	176	20	51	18	165	32	30	62	210	20	49	57	199	38	28	34	220	21	55	19	199	64	30	31	215
1011101	895	2	2	1	3	2	7	3	40	2	4	2	3	1	2	1	6	2	2	3	3	1	1	2	7	2	2	2	3	2	2	2	7
1011110	896	2	2	2	39	19	16	2	54	2	2	2	46	17	13	2	88	2	2	1	40	19	15	1	95	2	2	2	39	19	15	2	53
1011110	897	2	2	1	61	2	2	2	35	2	2	2	36	2	3	1	37	2	2	2	35	2	3	2	41	2	2	2	77	2	2	2	35
1011110	898	2	2	2	44	1	2	2	104	2	2	2	78	2	2	2	51	2	1	1	81	2	2	2	79	2	4	2	45	2	2	1	47
1011111	899	7	5	1	6	5	5	1	7	6	7	1	7	6	6	2	8	7	5	1	8	6	6	2	7	9	6	1	8	7	6	1	8
1011111	900	5	5	1	15	8	5	1	7	6	5	1	7	8	5	1	8	6	5	1	7	8	38	2	8	7	6	2	8	7	6	1	8
1100000	901	445	366	1	8043	498	356	1	7611	531	414	1	7596	571	428	1	7744	530	404	1	7664	542	440	2	7892	569	445	1	7652	565	459	1	7762
1100001	902	749	602	681	9687	720	668	633	9423	802	659	673	9619	791	682	689	9541	872	767	737	9565	893	746	794	9473	906	816	821	9766	925	827	840	9545
1100001	903	80	106	72	2230	91	85	144	2245	84	130	76	2250	128	122	102	2240	90	86	121	2277	112	164	131	2290	96	94	95	2237	125	110	120	2284
1100001	904	610	464	479	9015	614	472	507	8968	659	508	553	8985	655	530	518	9075	2426	2371	2336	10637	2484	2368	2426	10783	2630	2504	2528	10867	2719	2586	2605	10847
1100011	905	419	361	349	7837	478	371	333	7938	483	429	388	8036	539	426	406	7852	505	398	442	7749	517	440	412	7812	552	433	413	8075	585	422	426	7885
1100011	906	536	389	441	8797	525	432	409	8386	587	471	506	8650	575	487	460	8550	597	467	498	8619	605	507	513	8479	648	561	546	8435	706	519	525	8674
1100011	907	436	331	315	7828	444	348	364	7860	481	348	355	7689	526	389	402	7724	502	403	369	7926	512	398	435	7649	546	430	404	7919	541	410	444	7921
1100101	908	624	519	528	8826	641	479	482	8830	618	573	524	8974	656	558	554	9125	659	536	541	9000	668	559	555	8871	700	578	615	9000	726	629	598	8896
1100101	909	131	58	47	182	95	92	61	167	85	49	70	154	101	64	65	169	145	49	51	161	107	66	66	185	126	89	69	164	170	69	71	184
1100101	910	33	22	23	83	52	35	35	96	39	24	28	83	84	35	46	122	42	25	26	87	56	38	79	132	43	25	27	112	59	38	39	101
1100110	911	7	7	7	148	21	21	22	157	6	6	7	113	19	19	21	153	6	6	7	171	22	42	27	157	6	7	8	114	21	20	22	123

continued on next page

Table A.5: *XMarkAuto* absolute (ms): average over all iterations of a query with DB intervals
(continued from previous page)

class	ID	+DB intervals																															
		+indirect keyword occurrences																−indirect keyword occurrences															
		+index intervals								−index intervals								+index intervals								−index intervals							
		+label index				−label index				+label index				−label index				+label index				−label index				+label index				−label index			
		ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG
		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
1100111	912	502	336	321	7977	451	360	357	7933	491	355	361	7920	500	396	400	8041	507	380	399	7934	503	409	386	8054	559	438	462	7966	612	439	445	8013
1100111	913	434	316	349	7825	486	333	358	7721	467	388	366	7779	501	375	406	7903	512	405	388	7769	492	395	448	7768	524	465	439	7836	566	448	452	7688
1100111	914	442	361	367	7713	450	364	371	7769	497	390	405	7779	499	403	419	7859	505	380	417	7818	526	449	430	7955	534	470	455	7846	611	493	467	7986
1101001	915	364	377	2	488	453	414	1	517	423	391	1	484	485	445	2	488	1849	1751	2	9366	1897	1800	2	9300	1983	1850	3	9446	2058	1947	2	9511
1101001	916	426	381	7	445	455	413	2	456	423	392	1	458	493	454	2	544	1976	1863	2	9508	2029	1956	2	9556	2088	2015	1	9592	2270	2120	2	9587
1101001	917	493	339	2	7749	440	356	3	7648	571	388	2	7730	538	452	1	7722	2023	1852	2	16688	2119	1927	2	16627	2193	1998	2	16830	2286	2060	1	16717
1101010	918	2	37	2	112	42	47	2	91	2	2	2	112	16	16	2	125	20	11	1	1543	45	58	1	1569	23	13	2	1622	37	55	4	1543
1101010	919	87	85	2	620	148	117	1	667	96	78	2	558	117	95	1	661	14327	14359	2	22486	14971	15014	2	23101	15546	15556	1	23816	16379	16125	2	24343
1101011	920	440	382	2	470	430	459	2	501	398	393	2	475	489	453	2	469	936	852	2	8329	1014	902	1	8349	1021	982	3	8512	1112	973	1	8470
1101011	921	418	382	2	530	423	393	1	450	464	430	2	445	458	454	2	507	939	864	2	8423	1017	880	3	8292	1016	962	2	8548	1131	959	1	8487
1101011	922	435	407	2	440	431	390	1	479	455	447	2	467	455	419	1	478	971	851	2	8357	983	892	2	8455	1078	922	2	8405	1103	989	2	8640
1101100	923	66	47	1	1181	85	61	1	1202	50	53	1	1168	101	66	1	1235	52	49	2	1192	74	68	1	1239	83	54	1	1141	110	73	2	1211
1101100	924	95	109	111	980	99	102	151	1009	120	83	113	971	110	126	134	1054	133	85	122	993	133	130	101	1007	95	92	91	913	139	124	138	1050
1101100	925	44	11	10	11	15	16	17	16	11	15	11	11	17	16	17	16	11	43	13	11	19	18	74	51	11	11	17	47	13	46	13	17
1101101	926	28	23	31	54	41	36	35	94	31	26	31	56	68	37	70	93	33	31	28	88	45	94	40	67	35	34	31	60	46	49	41	67
1101110	927	26	29	28	566	43	42	42	556	31	31	31	566	44	42	44	562	33	30	67	558	78	47	48	561	35	34	39	472	100	47	48	608
1101111	928	3	3	4	5	50	50	23	53	3	23	5	5	48	18	20	22	3	3	4	6	20	20	22	59	5	5	4	6	18	19	20	22
1101111	929	64	41	49	273	67	60	62	271	76	49	54	226	68	69	67	261	52	85	50	263	74	96	98	281	64	54	75	235	77	101	69	247
1110001	930	500	399	435	8094	556	439	474	7941	579	531	467	7996	623	450	475	8099	2253	2151	2213	9661	2385	2221	2246	9874	2454	2339	2360	9802	2544	2413	2453	10044
1110001	931	71	22	22	1925	40	23	23	1781	35	23	32	1736	34	26	27	1787	35	25	24	1763	36	23	26	1735	36	25	58	1928	39	27	49	1799
1110001	932	29	22	24	1963	31	23	25	1752	32	25	24	1895	38	43	25	1808	35	25	46	1752	34	56	24	1763	41	24	26	1970	46	25	27	1762
1110010	933	181	41	23	2335	129	22	54	2237	130	23	25	2288	164	22	24	2183	624	370	443	7719	599	409	386	7419	667	441	418	7711	681	446	459	7496
1110011	934	29	24	20	1841	36	19	23	1782	69	22	25	1785	35	24	25	1763	33	24	24	1713	33	23	26	1694	36	25	28	1793	37	27	28	1666
1110011	935	534	338	351	8923	482	352	368	8972	500	408	411	8828	533	397	427	8915	562	470	447	9093	556	510	520	9121	613	537	551	9164	675	518	522	9077
1110100	936	70	21	24	1903	82	23	24	1897	73	22	24	1928	54	24	25	1869	54	22	23	2045	110	25	26	1981	57	26	27	1897	55	35	28	1967
1110100	937	92	22	44	1994	50	23	24	1912	52	24	26	1892	54	23	25	1798	83	24	23	2016	53	23	24	1961	87	24	26	1918	89	25	26	1878
1110101	938	33	20	22	1780	65	21	23	1788	67	54	26	1837	65	31	27	1776	33	24	26	1958	34	24	25	1851	40	24	26	1757	38	27	26	1819
1110101	939	30	40	24	1864	30	21	22	1812	33	25	26	1794	36	25	27	1778	35	23	28	1959	44	25	27	1851	36	24	48	1870	37	25	29	1767

continued on next page

Table A.5: *XMarkAuto* absolute (ms): average over all iterations of a query with DB intervals
(continued from previous page)

class	ID	+DB intervals																															
		+indirect keyword occurrences																−indirect keyword occurrences															
		+index intervals								−index intervals								+index intervals								−index intervals							
		+label index				−label index				+label index				−label index				+label index				−label index				+label index				−label index			
		ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG
		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
1110101	940	31	28	24	1853	42	24	25	1756	36	25	24	1817	34	25	26	1749	33	32	23	1934	34	25	26	1850	38	26	26	1833	41	25	28	1813
1110110	941	2	2	2	73	3	2	2	98	2	2	2	72	2	2	2	40	2	2	1	96	2	2	2	40	4	2	2	40	2	2	2	44
1110110	942	2	2	1	37	2	2	1	38	2	2	2	69	1	2	2	88	2	2	3	37	2	2	2	68	2	2	1	94	2	2	1	40
1110110	943	5	7	2	8	37	5	2	7	6	38	2	9	6	5	1	9	6	6	1	10	6	6	1	10	8	6	2	10	7	6	2	10
1110111	944	29	18	54	1805	31	20	21	1725	35	23	25	1794	58	23	25	1782	33	30	25	1905	34	45	26	1794	37	24	26	1996	41	27	26	1736
1110111	945	444	308	345	7667	460	328	335	7714	487	390	370	7714	503	393	411	7844	542	417	407	7791	518	382	410	7849	579	439	466	9700	545	416	454	8103
1110111	946	33	21	22	1758	32	20	23	1731	65	22	22	1753	32	24	23	1782	33	24	26	1769	34	25	26	1776	35	26	26	1826	39	27	28	1801
1111000	947	19	32	1	1480	20	11	1	1471	21	44	1	1459	22	14	1	1466	23	12	1	1455	53	13	2	1483	22	13	1	1565	36	14	1	1465
1111000	948	6	5	1	8	7	5	3	9	6	5	1	7	6	8	3	7	488	382	1	7688	489	426	1	7722	538	476	2	8603	586	430	1	7753
1111001	949	31	22	25	81	53	43	41	100	38	26	28	82	55	40	42	131	36	27	27	62	57	44	75	79	47	31	31	63	61	51	45	136
1111001	950	601	482	433	8089	591	509	505	8118	610	530	528	8247	596	512	535	7991	2311	2199	2275	9444	2361	2352	2337	9743	2542	2450	2421	9896	2688	2466	2495	9961
1111011	951	453	390	409	481	452	413	421	504	475	430	443	489	453	449	492	507	457	512	451	503	535	485	504	568	508	501	536	587	535	515	493	562
1111011	952	8	8	9	566	9	7	10	560	10	7	11	593	14	10	9	463	20	14	15	1338	19	12	15	1297	22	12	16	1370	21	13	14	1336
1111011	953	8	8	9	570	9	6	9	520	11	7	9	569	9	28	8	568	20	10	12	1365	19	13	12	1324	20	13	12	1320	20	14	14	1300
1111100	954	379	337	391	388	361	352	344	341	393	346	374	403	387	392	418	419	371	433	378	389	417	421	421	424	418	406	427	414	410	413	426	415
1111100	955	293	241	257	247	284	258	293	287	269	300	271	271	312	313	284	283	347	296	331	322	367	337	309	341	319	316	328	321	352	391	353	337
1111101	956	396	265	296	2703	378	275	285	2783	407	296	324	2726	405	357	336	2711	372	331	297	2723	378	306	301	2650	372	317	377	2785	441	349	377	2730
1111101	957	8	9	7	10	8	7	8	9	7	9	8	10	9	7	8	11	6	9	27	11	6	6	8	9	7	7	8	11	7	7	8	12
1111110	958	4989	4562	4785	4773	4769	4855	4877	5000	4848	5017	4987	5146	5120	5171	5241	5411	5343	5342	5507	5598	5513	5604	5602	5774	5693	5765	5720	5979	5948	5965	5903	6081
1111110	959	588	462	512	7662	562	487	475	7519	583	516	519	7636	614	549	534	7547	637	499	551	7642	656	542	529	7871	629	567	578	7877	621	590	592	7847
1111111	960	2	2	3	37	19	19	18	48	5	5	4	37	17	18	19	48	2	3	4	37	18	17	25	74	3	3	4	72	36	18	18	48
1111111	961	4	2	2	161	3	3	1	182	3	2	1	126	3	3	1	124	4	5	2	160	4	3	1	159	4	5	1	125	4	3	1	160
1111111	962	8	8	1	7	6	5	3	7	6	6	2	7	8	6	1	7	6	6	1	8	7	6	35	8	8	7	2	8	13	7	1	8

Table A.6: *XMarkAuto* absolute (ms): average over all iterations of a query without DB intervals

class	ID	−DB intervals																															
		+indirect keyword occurrences																−indirect keyword occurrences															
		+index intervals								−index intervals								+index intervals								−index intervals							
		+label index				−label index				+label index				−label index				+label index				−label index				+label index				−label index			
		ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG
		32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
0000011	800	1750	1432	1454	6972	1808	1474	1494	7396	1943	1612	1625	7275	1942	1611	1633	7160	1975	1661	1652	7406	1993	1698	1687	7388	2316	1742	1805	7089	2135	1770	1802	7763
0010100	801	3379	3404	3410	3424	3456	3466	3513	3666	3638	3661	3655	3700	3701	3754	3776	3762	3769	3776	3853	3862	3923	3846	3928	3938	4475	3983	3971	3968	4009	4063	3977	4097
0010100	802	48	52	49	47	49	53	50	50	52	53	53	51	54	60	67	53	54	55	59	58	60	93	61	58	65	62	87	56	58	63	63	58
0010110	803	7985	7898	7994	10018	8183	8116	8215	10543	8814	8757	8704	10763	9035	8902	8937	11004	9142	9145	9256	11216	9431	9350	9446	11457	11083	9526	9629	11609	9801	9784	9820	11940
0011011	804	608	468	476	2386	615	507	490	2402	663	485	490	2409	667	502	558	2397	681	577	585	2478	700	589	602	2516	791	645	630	2550	746	623	610	2601
0011100	805	277	253	256	259	261	261	327	305	331	312	282	301	346	317	290	288	318	319	322	315	354	352	312	321	473	306	302	308	369	365	340	339
0011100	806	19	19	20	19	20	20	21	20	23	21	24	21	24	22	22	22	24	30	22	22	22	32	23	22	25	25	25	22	25	25	25	23
0011100	807	10	10	10	9	10	10	10	10	12	10	10	10	12	10	10	10	13	11	11	11	11	13	11	11	20	11	11	11	11	11	13	11
0011100	808	253	289	292	285	270	262	268	299	287	309	337	280	283	310	325	313	296	322	323	352	328	325	304	319	372	337	363	307	314	337	348	314
0011100	809	12	10	10	10	10	10	10	42	12	11	10	9	10	11	12	9	11	10	11	9	11	11	11	10	11	11	11	10	11	11	13	10
0011110	810	2498	2547	2552	2608	2610	2607	2528	2678	2708	2705	2718	2805	2777	2786	2810	2824	2826	2910	2863	2895	2873	2871	2902	2947	3539	2942	2968	2956	2995	2958	3008	3105
0011111	811	4328	4382	4425	4488	4472	4523	4585	4771	4765	4801	4832	4861	4890	4937	4993	5036	5021	5040	5084	5168	5161	5178	5250	5295	6240	5315	5386	5395	5392	5414	5466	5553
0011111	812	1631	1651	1634	1691	1696	1733	1698	1772	1791	1792	1839	1845	1877	1886	1883	1936	1950	1887	1937	1957	1951	1947	2014	1988	3135	1996	2022	2051	2028	2038	2041	2111
0100011	813	629	456	461	2511	613	502	499	2542	627	565	530	2527	662	518	544	2661	1208	1156	1135	3184	1257	1156	1149	3156	2000	1163	1194	3153	1352	1198	1185	3219
0100100	814	12818	12848	13015	13175	13214	13245	13438	13873	14005	13990	14118	14074	14179	14296	14436	14479	14607	14637	14728	14892	14925	15013	15184	15165	18043	15466	15414	15442	15504	15570	15621	15783
0100101	815	1702	1606	1635	3039	1756	1662	1701	3207	1902	1818	1817	3323	1979	1866	1860	3327	1957	1861	1889	3286	2010	1936	1916	3317	2226	1962	1999	3369	2106	2002	2010	3384
0100101	816	5140	5089	5175	6903	5347	5288	5231	7235	5665	5623	5691	7781	5818	5784	5776	7851	5988	5875	5959	7610	6181	6067	6083	7887	6338	6255	6261	8031	6488	6414	6395	8192
0100101	817	589	473	500	2461	588	507	565	2391	643	538	545	2573	712	556	598	2467	691	515	529	2426	652	541	556	2505	670	571	592	2477	689	582	551	2427
0100110	818	591	472	504	2673	594	507	516	2570	692	573	562	2661	697	569	574	2684	648	526	558	2564	636	601	613	2668	690	591	587	2632	718	598	609	2587
0100111	819	558	454	472	2390	560	491	483	2307	591	520	524	2388	630	528	545	2379	633	495	540	2363	645	534	546	2428	665	568	561	2373	673	578	593	2404
0100111	820	546	445	478	2364	590	481	486	2313	627	528	524	2413	643	539	543	2441	635	498	531	2392	646	544	539	2402	677	558	563	2366	699	570	583	2342
0100111	821	1162	943	957	4849	1166	968	1025	4822	1278	1037	1039	4822	1309	1059	1057	4835	1294	1050	1084	4910	1317	1060	1076	4952	1376	1144	1150	4842	1426	1166	1157	4907
0100111	822	603	511	518	2470	627	520	527	2432	635	556	535	2444	661	543	548	2481	683	567	574	2500	693	578	581	2483	722	609	603	2582	728	613	616	2438
0101001	823	82	85	85	107	106	108	102	129	90	88	89	114	113	129	108	139	105	103	107	120	124	149	142	166	141	109	145	126	160	125	154	166
0101011	824	1447	1481	1471	1532	1514	1514	1521	1679	1593	1598	1643	1669	1685	1695	1673	1758	1710	1681	1733	1783	1792	1768	1810	1886	1844	1774	1786	1878	1851	1877	1845	1952
0101100	825	3356	3408	3455	3540	3488	3522	3537	3761	3647	3686	3684	3778	3859	3807	3816	3932	3827	3832	3866	3969	3888	3923	3932	4094	3985	3989	4044	4135	4100	4052	4075	4237
0101100	826	6066	6033	6004	6074	6127	6268	6093	6439	6390	6463	6374	6561	6591	6648	6517	6741	6589	6618	6802	6857	6901	6899	6905	6994	6916	6945	7056	7120	7181	7184	7174	7237
0101100	827	3331	3312	3338	3392	3358	3375	3423	3520	3527	3553	3513	3607	3621	3662	3649	3708	3637	3688	3826	3742	3857	3853	3825	3813	3841	3867	3872	3850	3918	3951	4042	3977

continued on next page

Table A.6: *XMarkAuto* absolute (ms): average over all iterations of a query without DB intervals
(continued from previous page)

class	ID	−DB intervals																															
		+indirect keyword occurrences																−indirect keyword occurrences															
		+index intervals								−index intervals								+index intervals								−index intervals							
		+label index				−label index				+label index				−label index				+label index				−label index				+label index				−label index			
		ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG
		32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
0101101	828	2	3	4	4	17	43	41	23	2	3	4	5	18	16	20	57	8	3	4	5	17	17	21	23	3	3	4	5	19	16	20	58
0101110	829	3364	3453	3464	3515	3467	3550	3481	3644	3673	3628	3682	3682	3744	3740	3794	3782	3800	3803	3964	3842	3915	3970	4043	3997	4012	3956	4037	4001	4122	4014	4193	4101
0101110	830	15	15	16	24	32	27	34	45	16	16	24	29	69	33	31	39	17	17	19	28	32	32	34	51	18	18	25	74	34	36	35	42
0101111	831	3	3	4	3	17	19	19	19	3	3	4	2	16	17	17	18	2	3	4	3	16	17	19	14	3	3	6	3	16	17	18	17
0101111	832	3	5	4	4	17	15	17	15	3	3	4	3	18	51	18	17	3	5	4	3	19	28	19	20	3	3	4	3	15	16	51	17
0101111	833	3	3	5	3	15	15	17	18	3	3	4	3	16	16	16	16	3	3	4	3	17	19	55	41	3	3	4	3	18	17	18	17
0101111	834	4	6	6	4	17	40	18	20	4	6	5	5	17	16	15	19	4	4	5	5	17	17	20	18	4	4	6	5	17	19	19	20
0101111	835	6	5	8	6	16	18	19	23	5	6	7	7	20	21	39	22	8	6	7	7	27	20	22	21	6	6	8	7	18	20	31	20
0101111	836	9	9	11	9	21	31	25	24	9	10	11	10	25	23	25	78	10	12	11	13	26	27	26	65	13	11	12	12	26	25	28	25
0101111	837	3	3	4	3	17	17	18	20	3	3	4	3	46	15	16	17	4	3	4	3	15	17	20	19	3	3	4	3	16	16	18	16
0110001	838	3469	3294	3391	5660	3517	3436	3492	5889	3729	3650	3688	6096	3868	3746	3757	6120	6003	5786	5867	9516	6096	5933	5981	9363	6309	6040	6101	9732	6405	6196	6251	9906
0110001	839	96	93	96	234	102	98	100	280	127	106	105	274	112	108	132	215	299	314	302	562	303	310	337	575	336	306	338	602	316	346	316	551
0110011	840	59	41	42	662	83	41	42	656	59	79	45	731	96	50	47	700	642	551	499	2815	643	495	527	2705	682	552	579	2726	716	557	564	2786
0110101	841	51	49	51	64	55	53	73	97	55	54	63	69	61	79	59	70	62	58	83	72	60	89	59	97	60	60	62	73	60	59	61	75
0110110	842	5881	5934	5949	6388	5943	5986	6034	6809	6295	6370	6303	6763	6426	6487	6454	7037	6560	6611	6590	7016	6697	6702	6782	7261	6805	6732	6927	7342	6970	7000	7024	7438
0110111	843	7	3	5	11	3	3	5	11	3	4	8	11	3	4	4	11	3	7	5	14	4	3	5	11	4	4	5	14	4	4	5	19
0110111	844	3	3	4	53	3	3	4	53	4	4	5	15	4	3	5	18	3	3	5	14	4	3	5	19	3	3	5	53	3	4	5	17
0110111	845	3	3	40	13	3	3	5	13	4	3	5	11	3	4	5	13	4	3	5	14	29	3	5	38	3	4	25	17	6	3	7	51
0110111	846	6663	6722	6764	7295	6737	6831	6814	7649	7166	7236	7343	7641	7378	7411	7343	7932	7594	7570	7638	8031	7710	7706	7762	8147	7890	7830	7979	8361	8014	8062	8079	8594
0111000	847	4333	4300	4362	4610	4537	4431	4468	4889	4645	4659	4711	5005	4781	4825	4829	5113	5189	5215	5264	5437	5342	5333	5378	5590	5478	5436	5481	5741	5561	5560	5570	5827
0111001	848	6828	6815	6882	7290	7000	7037	7115	7641	7307	7334	7376	7821	7517	7498	7456	7968	7624	7657	7736	8031	7811	7902	7873	8271	7993	8036	8037	8438	8124	8148	8217	8524
0111001	849	10	10	11	16	10	10	15	17	10	11	12	19	11	13	14	20	30	18	15	20	18	13	17	20	19	14	15	21	27	14	17	29
0111001	850	229	255	233	250	307	236	238	250	307	310	252	281	275	256	256	296	297	283	290	335	341	290	357	312	336	332	300	320	318	327	306	388
0111100	851	9	9	9	9	9	9	9	9	10	9	46	9	12	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	11	10	10	10
0111100	852	245	274	300	240	242	267	244	279	257	260	284	297	265	291	288	304	296	333	335	288	302	302	280	305	282	309	311	323	322	297	327	321
0111110	853	1648	1679	1691	1745	1666	1716	1701	1800	1824	1846	1817	1806	1816	1868	1805	1843	1839	1845	1866	1975	1900	1895	1955	1922	1914	1917	1923	1952	1973	1997	1965	1979
1000001	854	1665	1467	1518	3922	1704	1503	1509	3946	1817	1605	1614	3959	1881	1637	1707	4082	1913	1713	1694	4112	1887	1714	1729	4188	2003	1788	1781	4189	2045	1827	1837	4194
1000001	855	791	663	677	2940	821	671	694	2845	873	745	728	2852	902	727	732	2826	889	717	717	2882	873	733	739	2973	920	773	774	2948	938	766	767	2990

continued on next page

Table A.6: *XMarkAuto* absolute (ms): average over all iterations of a query without DB intervals
(continued from previous page)

class	ID	−DB intervals																															
		+indirect keyword occurrences																−indirect keyword occurrences															
		+index intervals								−index intervals								+index intervals								−index intervals							
		+label index				−label index				+label index				−label index				+label index				−label index				+label index				−label index			
		ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG
		32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
1000010	856	621	422	453	2681	644	462	455	2636	651	469	520	2728	673	499	506	2572	662	471	505	2647	676	545	512	2765	702	535	534	2747	707	539	542	2762
1000011	857	568	419	440	2518	586	443	485	2379	633	465	491	2504	633	487	466	2403	631	465	492	2442	640	501	512	2509	678	550	526	2526	681	523	526	2574
1000011	858	1119	845	851	5136	1185	895	874	4975	1265	973	992	5072	1279	1007	963	4922	1487	1362	1375	3398	1535	1392	1431	3486	1545	1429	1442	3534	1600	1492	1517	3575
1000011	859	664	475	482	2830	637	475	455	2781	659	489	516	2872	699	520	499	2754	732	602	608	2866	741	590	620	2958	784	647	680	2956	786	650	653	2953
1000101	860	910	804	770	3135	927	802	791	3049	983	860	840	3144	1035	881	852	3081	1018	883	858	3175	1012	840	847	3224	1075	880	860	3221	1095	896	926	3209
1000101	861	796	683	662	2882	807	656	684	2826	859	723	726	2873	885	729	775	2815	868	719	759	2860	880	729	725	2963	977	761	756	2982	958	772	809	3002
1000101	862	592	473	512	2494	640	486	491	2455	644	515	518	2542	657	526	528	2429	676	513	517	2529	653	520	525	2609	701	540	544	2620	694	583	549	2642
1000111	863	546	431	438	2391	574	421	453	2426	653	519	479	2460	657	523	497	2386	637	486	467	2453	668	532	491	2519	677	510	487	2517	679	527	530	2561
1000111	864	565	442	440	2452	617	414	477	2453	623	477	478	2493	645	485	485	2498	633	479	485	2481	667	489	494	2546	668	542	487	2554	693	539	520	2579
1000111	865	607	429	459	2534	553	406	436	2430	629	440	497	2528	606	479	485	2463	642	467	464	2484	640	478	491	2552	680	500	476	2594	710	546	512	2590
1001100	866	5	5	6	6	19	20	17	19	5	6	7	5	37	16	19	17	12	12	45	5	19	17	19	25	5	8	5	6	16	17	16	17
1001100	867	5	10	7	5	68	10	15	13	5	3	3	5	16	11	9	12	4	4	4	5	14	24	13	46	5	5	3	5	9	13	9	13
1001101	868	1	1	2	1	18	18	19	17	1	1	3	2	15	16	51	15	1	1	4	1	15	18	18	17	1	1	3	1	47	15	18	47
1001101	869	6064	6037	2	6413	6205	6205	1	6715	6589	6521	1	6818	6638	6720	2	6974	6870	6806	1	7062	7025	6930	1	7196	7007	7041	2	7349	7203	7235	2	7467
1001110	870	596	480	2	2405	652	458	2	2331	620	497	2	2361	699	505	1	2307	656	499	2	2441	670	544	2	2491	703	503	2	2460	716	549	2	2504
1001111	871	595	470	2	2319	570	505	1	2369	661	545	2	2291	637	525	1	2446	623	559	2	2267	643	527	1	2327	673	552	2	2438	712	556	2	2372
1010001	872	813	655	672	2730	785	668	699	2877	865	713	737	2865	868	728	765	2986	1002	841	859	3004	1006	836	866	2917	1090	866	949	3134	1068	868	878	2998
1010011	873	684	555	532	3486	711	534	557	3414	765	583	588	3507	795	624	589	3547	1957	1611	1651	7491	1984	1674	1653	7689	2079	1711	1726	7684	2109	1769	1776	7414
1010011	874	667	516	534	3152	706	529	532	3209	724	544	564	3261	745	585	565	3220	1952	1624	1674	7339	1980	1632	1697	7560	2119	1749	1718	7552	2112	1751	1793	7316
1010011	875	563	435	433	2458	588	436	425	2474	632	454	456	2487	634	454	529	2507	627	485	514	2451	639	518	488	2552	678	531	513	2552	678	511	526	2407
1010101	876	1402	1293	1303	3488	1448	1286	1333	3560	1566	1428	1447	3641	1602	1436	1449	3665	1581	1443	1467	3587	1636	1509	1486	3681	1702	1549	1559	3728	1729	1574	1563	3700
1010101	877	794	645	661	2825	837	655	664	2915	864	732	705	2963	892	747	755	2977	872	728	735	2878	862	714	723	2944	910	745	773	2969	942	745	761	2928
1010110	878	7	8	2	8	7	6	1	8	7	7	1	9	7	7	2	9	8	7	2	28	8	7	2	9	8	7	2	9	8	7	2	9
1010110	879	9	6	2	8	7	6	1	8	9	7	2	8	7	7	1	9	8	7	1	8	10	7	3	9	8	7	2	9	8	7	2	9
1010111	880	594	431	2	2368	597	430	2	2382	651	461	2	2468	604	446	2	2507	633	480	2	2401	608	480	2	2480	690	502	2	2461	669	489	2	2407
1010111	881	567	406	436	2439	620	445	428	2476	625	447	479	2483	628	465	487	2493	661	477	483	2442	682	463	493	2503	666	516	514	2496	672	488	527	2398
1010111	882	576	417	2	2403	587	415	2	2434	632	453	2	2510	643	457	2	2489	654	548	2	2401	655	582	3	2449	714	574	2	2477	692	557	2	2445
1011000	883	558	456	2	2459	599	459	2	2416	635	493	2	2496	677	515	2	2506	642	530	2	2490	663	542	1	2506	682	572	2	2516	708	572	2	2449

continued on next page

Table A.6: *XMarkAuto* absolute (ms): average over all iterations of a query without DB intervals
(continued from previous page)

class	ID	−DB intervals																															
		+indirect keyword occurrences																−indirect keyword occurrences															
		+index intervals								−index intervals								+index intervals								−index intervals							
		+label index				−label index				+label index				−label index				+label index				−label index				+label index				−label index			
		ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG
		32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
1011000	884	2537	2647	1	2669	2674	2610	1	2810	2750	2770	1	2851	2817	2816	2	2934	3498	3391	1	5426	3572	3539	2	5503	3752	3659	3	5520	3713	3609	2	5578
1011001	885	7	6	1	8	9	6	2	8	10	6	2	9	7	46	2	9	621	490	1	2328	628	508	2	2356	663	522	2	2377	664	556	2	2352
1011001	886	616	542	555	2779	656	565	559	2866	700	591	584	2898	736	593	631	2874	2064	1938	1994	3916	2094	1993	2025	4147	2179	2035	2093	4226	2190	2068	2133	4063
1011001	887	562	558	586	629	580	531	569	633	639	620	580	670	619	605	630	686	643	660	662	698	646	677	640	702	701	648	696	713	710	667	701	693
1011010	888	7	6	1	8	7	6	1	8	8	7	2	8	8	7	2	8	657	495	2	2349	637	526	2	2380	670	557	2	2457	671	557	2	2396
1011010	889	7	6	2	8	7	6	2	8	8	7	2	9	7	8	2	9	630	478	2	2303	631	553	2	2475	658	566	2	2447	673	550	2	2374
1011011	890	370	291	320	873	377	295	299	910	429	353	379	926	447	346	339	875	438	347	353	945	425	330	396	936	493	354	347	982	446	349	418	946
1011011	891	562	490	2	2403	596	470	2	2399	628	526	2	2357	644	547	2	2432	655	531	2	2375	658	535	2	2523	684	533	2	2512	698	565	2	2449
1011011	892	1739	1649	1687	3684	1806	1700	1757	3878	1911	1807	1811	3849	1934	1832	1860	3921	1985	1874	1908	3839	2108	1950	1937	3995	2084	1991	1978	3975	2155	1967	1988	4013
1011100	893	9	8	11	222	26	27	56	230	10	10	9	240	58	21	23	217	10	8	11	223	25	56	23	203	11	8	10	158	26	21	29	253
1011100	894	12	9	48	55	29	40	30	109	13	10	18	87	28	26	28	64	13	10	10	56	29	27	26	90	20	11	50	59	29	62	26	65
1011101	895	2	2	2	2	2	2	2	4	2	2	9	4	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	3	2	2	2	2
1011110	896	2	2	2	8	19	15	2	24	2	2	1	11	17	13	2	60	4	2	2	8	16	15	2	62	2	2	1	9	17	33	2	27
1011110	897	2	2	2	6	2	2	2	7	2	2	2	14	3	2	2	14	2	2	2	6	2	2	2	7	2	2	2	7	2	2	2	7
1011110	898	2	2	2	49	2	2	2	8	2	2	2	13	2	2	2	11	2	2	2	50	2	2	2	15	2	2	2	13	2	2	2	48
1011111	899	7	6	1	8	7	6	2	8	39	7	2	8	9	7	2	10	8	6	2	8	8	7	2	9	8	7	2	9	11	7	1	9
1011111	900	7	6	2	8	9	6	2	8	7	7	2	8	8	7	2	9	8	6	1	8	10	7	2	9	10	7	2	9	8	7	2	9
1100000	901	620	445	1	2389	627	510	2	2459	633	490	2	2391	647	533	2	2390	659	535	1	2368	660	510	2	2452	688	586	1	2612	706	590	2	2435
1100001	902	903	780	793	3135	903	776	772	3200	1015	827	887	3226	1000	886	862	3314	1051	889	898	3117	1064	897	908	3202	1069	936	917	3227	1095	936	945	3258
1100001	903	870	916	887	1561	942	925	925	1642	968	975	991	1608	1008	1029	1019	1608	1053	1047	1064	1648	1089	1070	1080	1724	1079	1106	1115	1828	1124	1129	1147	1744
1100001	904	740	560	566	3119	725	580	627	3149	792	579	634	3114	848	684	635	3147	2031	1865	1872	4041	2069	1960	1996	4199	2135	1978	1994	4263	2148	2007	2083	4253
1100011	905	564	486	450	2370	596	464	459	2445	625	475	493	2470	635	499	479	2504	642	469	513	2373	641	505	509	2476	644	526	561	2524	709	571	538	2549
1100011	906	653	535	578	2579	661	577	550	2683	747	587	586	2742	750	605	601	2655	723	583	598	2638	758	619	624	2705	765	643	642	2748	789	653	690	2786
1100011	907	590	447	442	2431	581	419	478	2465	617	453	513	2421	602	487	488	2447	634	489	466	2354	644	524	528	2422	706	525	504	2497	679	523	505	2522
1100101	908	1423	1328	1339	3472	1493	1333	1345	3638	1554	1451	1428	3616	1592	1453	1493	3669	1629	1512	1496	3612	1633	1518	1494	3648	1685	1535	1546	3766	1718	1590	1602	3852
1100101	909	117	79	81	132	159	90	91	151	128	80	107	140	203	96	97	184	155	86	83	145	205	122	133	163	138	96	87	191	154	126	134	173
1100101	910	78	26	30	59	61	40	41	104	49	30	32	99	64	72	43	73	49	30	32	64	66	83	45	109	55	30	33	63	98	45	45	79
1100110	911	6	7	8	41	20	22	25	48	8	8	8	41	23	21	55	48	7	8	9	41	22	22	25	48	8	8	9	34	22	56	23	49

continued on next page

Table A.6: *XMarkAuto* absolute (ms): average over all iterations of a query without DB intervals
(continued from previous page)

class	ID	−DB intervals																															
		+indirect keyword occurrences																−indirect keyword occurrences															
		+index intervals								−index intervals								+index intervals								−index intervals							
		+label index				−label index				+label index				−label index				+label index				−label index				+label index				−label index			
		ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG
		32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
1100111	912	566	447	420	2433	591	479	456	2517	662	484	509	2561	676	494	471	2528	667	490	527	2539	682	498	504	2512	674	521	501	2609	694	529	562	2603
1100111	913	603	432	430	2335	577	478	516	2476	622	493	539	2496	634	507	483	2420	640	506	503	2443	640	508	542	2448	708	528	527	2522	679	546	548	2594
1100111	914	569	455	469	2420	588	463	449	2555	631	500	502	2573	640	508	526	2481	608	505	508	2559	649	556	517	2488	692	535	540	2583	691	543	548	2555
1101001	915	328	304	2	332	371	296	2	348	335	360	2	355	339	377	1	378	2000	1895	2	3958	2077	1946	3	4003	2146	2024	2	4030	2179	2036	2	4144
1101001	916	310	277	2	347	341	305	2	396	313	334	2	327	356	329	2	412	2203	2057	2	4171	2254	2152	2	4209	2330	2189	2	4249	2361	2265	2	4327
1101001	917	579	448	2	2416	592	481	39	2439	647	492	1	2476	686	557	2	2425	2530	2311	1	6274	2559	2332	2	6281	2669	2379	2	6376	2702	2448	2	6452
1101010	918	1	2	2	22	18	15	1	77	1	2	2	23	17	16	2	71	27	17	2	467	73	28	2	496	26	18	2	463	43	28	2	509
1101010	919	109	131	1	261	166	139	2	274	142	123	2	273	169	139	1	297	16537	16587	2	19038	17076	17065	2	19574	17535	17484	2	19860	17954	17921	2	20371
1101011	920	353	278	2	303	334	304	2	335	336	309	2	325	367	354	2	345	996	847	2	2912	995	913	2	2939	1047	896	2	2986	1076	987	2	2992
1101011	921	290	347	2	330	325	371	2	364	314	361	2	358	410	387	2	353	955	831	2	2895	1004	853	2	2996	1044	875	1	2887	1096	922	2	2937
1101011	922	314	278	2	311	312	296	3	362	377	327	2	385	392	373	22	409	954	836	2	2863	1026	890	1	2976	1017	965	1	2966	1077	913	2	3050
1101100	923	65	55	1	292	82	107	1	345	71	68	1	338	86	111	1	366	66	71	1	360	108	81	2	347	71	66	1	332	123	111	2	314
1101100	924	81	84	108	324	95	98	97	380	126	90	121	337	136	97	147	352	91	89	91	339	134	129	99	325	124	87	92	330	105	102	103	387
1101100	925	10	11	42	12	18	46	18	15	17	17	17	11	15	15	17	15	44	43	12	12	18	18	50	18	12	11	15	11	17	17	15	15
1101101	926	37	30	32	47	49	43	43	54	40	33	36	43	53	43	48	55	61	38	35	47	52	46	50	89	43	38	39	47	53	45	47	58
1101110	927	64	37	38	131	51	50	50	141	40	42	42	189	53	53	83	152	47	39	42	161	55	84	56	165	45	62	50	134	58	56	58	147
1101111	928	3	3	5	4	21	20	22	20	3	4	5	7	18	69	20	22	4	3	5	5	20	40	22	54	3	3	5	7	19	19	21	56
1101111	929	60	55	57	102	78	74	129	119	66	69	63	108	83	75	79	122	67	124	63	107	86	78	112	123	71	103	70	112	90	82	83	123
1110001	930	664	539	546	2896	678	521	553	2928	755	571	571	3002	738	580	618	2963	1971	1805	1801	3916	1986	1876	1855	4037	2078	1873	1916	3956	2081	1926	2002	4136
1110001	931	39	28	27	539	72	27	28	521	42	30	30	514	44	31	32	549	44	27	56	516	44	28	35	548	44	31	31	518	49	32	40	555
1110001	932	42	28	30	546	39	27	30	510	44	29	31	535	42	29	33	506	42	28	30	520	44	28	30	512	46	31	33	489	44	31	38	510
1110010	933	77	27	30	781	85	28	29	828	110	27	31	780	115	28	31	789	678	529	508	2754	685	479	510	2682	714	500	506	2699	733	519	528	2624
1110011	934	62	25	28	508	82	28	27	558	40	29	31	533	44	30	30	467	41	30	33	493	46	62	32	506	44	52	64	471	49	31	88	513
1110011	935	601	447	453	2754	596	513	484	2799	656	525	500	2895	676	564	522	2855	729	535	598	2850	695	583	607	2858	725	631	595	2843	744	602	606	2741
1110100	936	51	28	50	578	83	27	61	516	62	30	30	622	54	31	32	616	53	50	31	622	54	28	32	587	82	30	31	581	61	30	32	544
1110100	937	51	46	29	577	51	26	30	617	105	31	31	543	59	31	32	605	85	34	31	538	79	30	32	558	81	29	33	580	65	30	40	609
1110101	938	38	33	30	543	41	28	29	574	42	30	32	536	47	29	33	546	41	30	31	519	45	31	30	514	46	32	33	543	45	30	34	509
1110101	939	40	29	28	544	41	29	28	537	46	28	32	518	43	29	39	535	45	30	29	519	42	28	30	533	45	30	31	541	48	32	34	550

continued on next page

Table A.6: *XMarkAuto* absolute (ms): average over all iterations of a query without DB intervals
(continued from previous page)

class	ID	−DB intervals															
		+indirect keyword occurrences															
		+index intervals								−index intervals							
		+label index				−label index				+label index				−label index			
		ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG
		32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
1110101	940	38	26	29	536	41	26	30	547	42	29	30	540	44	29	56	545
1110110	941	2	2	2	15	4	2	2	15	2	34	2	15	2	4	2	15
1110110	942	2	2	2	7	2	3	2	7	2	2	2	7	2	2	2	7
1110110	943	7	6	1	8	9	6	1	8	7	27	2	9	8	7	1	9
1110111	944	63	27	26	525	70	33	27	528	41	28	31	535	44	28	31	504
1110111	945	573	453	451	2512	556	483	440	2485	649	513	491	2495	642	495	498	2475
1110111	946	39	25	26	524	62	27	28	534	41	29	29	526	44	30	32	535
1111000	947	23	16	1	453	24	16	2	430	25	17	1	460	52	17	1	467
1111000	948	7	6	2	7	9	6	1	8	8	9	2	8	8	9	2	10
1111001	949	41	31	32	69	61	46	47	85	44	30	34	71	63	46	49	84
1111001	950	652	577	587	2891	688	559	560	2881	694	608	611	2941	734	585	632	2848
1111011	951	473	438	474	474	488	509	461	524	549	534	518	539	530	516	523	585
1111011	952	12	11	10	152	12	8	12	175	11	11	10	152	10	9	11	186
1111011	953	11	9	9	154	9	9	11	184	11	8	10	154	12	8	9	152
1111100	954	270	233	235	232	240	270	238	251	278	250	280	275	252	292	264	292
1111100	955	207	182	241	183	212	195	188	239	216	223	199	198	228	202	259	203
1111101	956	417	334	369	933	427	344	351	949	490	394	376	930	461	399	402	927
1111101	957	7	30	9	11	8	8	9	8	9	8	9	9	8	8	10	11
1111110	958	3399	3460	3460	3524	3520	3527	3558	3691	3631	3684	3724	3733	3689	3709	3785	3864
1111110	959	634	548	527	2478	646	563	540	2463	694	602	579	2568	675	607	580	2485
1111111	960	3	3	4	9	19	15	20	29	3	3	4	9	48	16	18	27
1111111	961	4	3	2	31	4	3	2	101	4	3	2	72	15	3	2	52
1111111	962	7	7	2	8	7	9	1	8	10	8	2	10	8	8	1	9

class	ID	−DB intervals															
		−indirect keyword occurrences															
		+index intervals								−index intervals							
		+label index				−label index				+label index				−label index			
		ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG
		48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
1110101	940	42	28	29	486	44	30	32	513	48	32	33	550	65	32	34	543
1110110	941	2	2	2	49	2	2	2	15	2	2	2	15	2	4	2	47
1110110	942	2	2	1	7	2	2	2	7	2	2	2	7	2	2	2	7
1110110	943	8	7	1	9	8	7	1	9	8	7	2	8	10	7	2	9
1110111	944	41	38	30	499	44	33	30	552	48	32	33	506	45	30	34	558
1110111	945	671	521	480	2520	640	498	539	2914	669	554	510	2555	681	534	508	2528
1110111	946	41	28	29	526	46	30	30	635	71	32	31	537	45	32	57	535
1111000	947	27	17	2	468	28	15	3	495	29	16	2	464	28	16	1	463
1111000	948	649	508	2	2435	686	495	1	2702	668	519	3	2445	701	529	1	2461
1111001	949	45	30	34	53	68	81	80	86	56	35	34	50	65	83	49	72
1111001	950	1898	1792	1840	3893	1977	1815	1847	4563	1968	1915	1926	3986	2077	1916	1957	4051
1111011	951	540	566	509	559	559	515	546	734	569	552	569	623	573	601	572	611
1111011	952	25	14	36	409	56	14	17	527	30	14	18	386	57	15	16	383
1111011	953	23	13	16	443	25	13	15	447	25	14	14	409	27	14	15	431
1111100	954	293	268	264	263	289	308	306	292	269	309	273	292	308	314	286	303
1111100	955	237	206	241	266	209	211	236	259	248	249	249	248	218	264	220	219
1111101	956	425	419	364	1003	439	396	433	1120	504	440	418	985	458	423	417	981
1111101	957	8	7	11	9	8	8	10	13	8	8	10	10	8	8	10	9
1111110	958	3797	3758	3881	3856	3903	3861	3868	4537	3924	3944	3991	4088	4034	3995	4073	4147
1111110	959	708	608	609	2562	717	621	591	2725	752	645	641	2508	762	637	649	2564
1111111	960	3	3	4	9	19	18	21	31	3	5	4	9	17	16	19	27
1111111	961	5	3	2	32	4	5	9	79	4	3	1	62	4	3	2	32
1111111	962	8	8	2	8	7	8	2	9	10	8	3	11	8	8	1	9

Table A.7: *NPMan* absolute (ms): average over all iterations of a query with DB intervals

class	ID	+DB intervals																															
		+indirect keyword occurrences																−indirect keyword occurrences															
		+index intervals								−index intervals								+index intervals								−index intervals							
		+label index				−label index				+label index				−label index				+label index				−label index				+label index				−label index			
		ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG
		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0001111	500	523	164	160	1508	527	229	243	1707	377	204	168	1488	518	224	214	1697	333	200	173	1455	536	225	226	1821	413	198	190	1495	562	213	246	1751
0001111	501	281	59	49	3758	457	91	120	3730	203	124	48	3450	413	81	97	3466	202	48	50	3487	428	91	132	3605	264	53	54	3469	457	96	98	3487
0100101	502	6337	5942	5953	34755	6332	5810	5905	34184	6405	6021	6061	34216	6732	6206	6292	34027	6941	6439	6480	33814	7123	6584	6574	34221	8776	6739	6860	35220	7503	6865	6970	34843
0100111	503	167	11	10	7155	215	45	27	7380	170	14	11	6837	236	19	32	6648	178	10	11	7602	303	19	21	7323	224	10	11	6980	279	30	20	6665
0100111	504	687	166	154	27404	661	151	154	27229	715	161	164	27212	729	157	159	27784	686	154	170	27151	730	182	179	27507	753	195	187	27756	792	199	185	27860
0100111	505	295	68	64	18699	257	63	69	19421	221	65	88	19453	284	69	82	18697	256	63	152	19008	313	72	86	19982	262	77	91	19718	313	77	94	18752
0100111	506	2116	336	297	67347	2013	342	313	66931	2166	347	313	67932	2286	307	401	68683	2236	338	322	66326	2296	316	337	68840	2502	394	358	69342	2547	377	404	68963
0100111	507	706	119	108	28312	576	159	113	27318	617	139	114	27403	664	111	123	26885	675	116	192	27011	718	141	130	27057	747	130	133	27182	777	126	132	26986
0101101	508	296	213	265	1279	317	234	272	1196	252	225	227	1081	361	240	250	1367	295	255	261	1136	350	251	309	1361	313	249	302	1300	358	292	296	1398
0101111	509	76	14	20	1225	120	26	34	924	58	15	17	786	154	32	42	1148	59	16	18	830	160	27	45	1089	66	17	24	1074	136	33	78	1153
0101111	510	23	11	13	75	103	20	29	110	31	25	19	58	83	21	22	131	27	13	35	70	88	22	98	139	29	14	16	75	92	22	31	134
0101111	511	52	16	17	1022	126	28	36	902	77	17	23	758	128	29	83	1155	59	18	24	810	159	30	55	1009	71	20	20	1066	131	30	36	1154
0110001	512	4	2	2	141	2	2	3	64	2	2	3	61	1	2	3	130	10422	8618	8691	75840	10697	8797	8916	75517	11474	9127	9278	77521	11461	9370	9379	78223
0110011	513	2	2	3	1	2	2	2	1	2	2	3	2	2	2	3	2	2218	342	354	64434	2363	350	379	65225	2478	393	407	65171	2524	349	389	65498
0110101	514	6901	6589	6805	7110	6560	6671	6657	7091	6722	6900	6809	7221	6963	6985	7076	7950	7208	7222	7304	7665	7453	7411	7514	7932	7628	7624	7682	8034	7731	7718	7877	8228
0110111	515	18	10	14	205	10	15	11	265	10	10	11	247	10	10	12	198	10	11	12	239	10	11	12	243	11	11	12	267	11	12	13	267
0111001	516	2	2	3	80	2	2	3	63	2	2	3	48	2	22	3	62	10437	8602	8587	75944	10689	8852	8899	74652	11201	9136	9306	76853	11427	9378	9445	75512
0111011	517	1	2	3	2	2	2	3	2	1	2	2	2	2	2	3	2	2252	333	350	64437	2273	318	344	64298	2471	342	365	66220	2532	357	376	64852
0111100	518	71	78	58	57	59	61	61	60	82	71	63	63	64	65	65	74	87	68	85	67	74	69	107	71	72	72	73	73	73	107	89	74
0111100	519	74	63	66	569	56	57	87	365	57	58	64	384	116	83	66	361	68	83	99	335	64	87	81	362	66	67	68	344	68	68	73	375
0111101	520	317	329	305	316	265	306	274	322	266	309	322	328	274	283	303	380	317	286	368	350	292	294	331	361	327	335	354	359	307	308	346	369
0111111	521	10	12	12	17	11	11	12	18	11	16	12	18	12	12	13	17	17	12	14	19	12	13	14	19	12	13	20	19	13	13	14	19
0111111	522	9	11	16	367	10	10	11	379	10	10	16	355	10	15	16	355	10	10	17	355	10	16	17	382	11	11	12	360	11	11	12	379
1001111	523	62	11	12	1369	177	37	29	1353	67	11	13	1284	180	25	31	1359	69	12	13	1189	163	26	27	1358	77	13	15	1553	188	26	42	1392
1001111	524	313	170	183	1383	524	263	237	1375	288	183	163	1105	530	192	200	1400	305	185	217	1171	565	204	246	1365	352	195	191	1140	594	229	219	1362
1001111	525	186	56	49	2558	439	99	95	2932	196	47	69	2539	444	82	132	2732	228	49	50	2772	459	85	98	3054	225	52	53	2546	447	86	99	3006
1011111	526	5	6	7	1204	4	5	7	1141	5	5	6	1456	5	5	6	1173	5	5	26	1117	5	6	7	1146	6	5	7	1134	5	6	7	1152
1100100	527	106	34	38	23	53	69	68	53	54	35	79	24	76	52	64	50	24	30	83	45	54	55	67	72	24	24	31	48	51	52	92	70

continued on next page

Table A.7: *NPMan* absolute (ms): average over all iterations of a query with DB intervals
(continued from previous page)

class	ID	+DB intervals																															
		+indirect keyword occurrences																−indirect keyword occurrences															
		+index intervals								−index intervals								+index intervals								−index intervals							
		+label index				−label index				+label index				−label index				+label index				−label index				+label index				−label index			
		ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG
		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
1100101	528	597	216	167	28824	665	160	163	27962	646	209	211	29241	690	176	213	27609	695	147	163	27708	716	196	196	27522	767	153	182	28476	801	184	184	27668
1100111	529	2168	307	300	66275	2015	287	315	65859	2144	352	321	68261	2216	336	325	65071	2240	354	355	66744	2356	338	334	67154	2534	371	435	64571	2546	350	394	67119
1100111	530	562	103	104	26842	601	110	149	26938	663	129	111	26755	692	131	115	27984	636	102	114	26986	681	117	191	26659	732	114	122	27169	738	144	128	28420
1100111	531	221	69	50	19835	274	69	63	18178	245	54	75	18591	268	62	78	17364	223	55	56	18486	334	64	78	18178	247	86	63	18630	300	69	97	17710
1100111	532	153	4	4	7002	220	27	14	7702	166	4	5	6815	250	13	14	6627	176	3	5	6756	233	13	14	7105	217	4	5	6814	273	12	18	6630
1100111	533	566	104	105	27147	624	116	121	27895	637	117	119	27200	720	118	168	28625	669	108	121	27780	676	114	125	27185	704	146	139	27438	766	125	133	27721
1100111	534	1942	356	1	64153	1973	330	2	65761	2130	419	2	66836	2200	379	1	64628	2271	354	2	67443	2292	388	1	65352	2461	398	1	67323	2588	425	1	63075
1101100	535	2	3	8	3	69	78	83	78	3	3	3	3	67	73	82	77	3	3	8	2	76	77	84	78	3	3	9	3	67	73	79	66
1101101	536	57	17	14	917	144	25	34	938	52	14	63	839	149	31	112	976	54	13	15	890	130	27	40	966	60	15	17	889	124	27	35	981
1101110	537	575	558	600	392	410	648	636	490	416	563	614	347	377	448	632	414	319	344	613	399	477	432	587	472	360	348	571	362	396	391	670	398
1101111	538	61	7	2	986	149	21	1	870	74	8	1	774	140	21	1	871	51	8	1	790	147	22	1	857	57	9	2	779	156	21	1	842
1101111	539	48	8	10	771	120	19	25	915	51	9	16	777	117	19	21	884	53	9	10	780	158	20	34	828	58	9	11	765	123	20	28	877
1101111	540	21	7	1	271	89	19	2	299	43	13	1	276	88	20	1	276	47	7	1	271	93	20	21	369	27	10	1	250	92	20	1	336
1101111	541	2	1	2	2	79	13	1	87	2	1	1	1	70	20	1	83	1	0	1	1	74	15	2	72	2	0	1	1	70	14	2	69
1101111	542	52	10	10	770	119	21	28	867	50	11	13	806	116	22	29	844	52	10	12	774	121	22	28	863	68	11	20	823	157	23	29	911
1101111	543	589	108	2	26714	576	128	1	28510	606	112	2	27120	669	131	1	27531	656	114	1	28671	693	130	2	27189	727	149	1	27809	731	164	1	27213
1101111	544	15	2	1	54	89	20	1	114	16	2	2	57	82	15	1	118	16	2	2	58	86	15	1	117	18	2	2	56	85	14	1	122
1101111	545	2	2	1	1	57	58	1	52	7	0	2	2	48	58	1	48	2	0	1	2	52	82	1	51	2	0	2	2	47	52	2	62
1101111	546	16	3	4	58	86	16	12	121	16	3	9	48	67	11	12	109	16	3	4	61	71	12	22	113	18	3	4	53	71	11	27	145
1110001	547	3	2	3	48	2	22	3	59	2	2	3	60	2	2	3	58	10435	8581	8619	78221	10704	8825	8901	74920	11197	9135	9211	77631	11394	9362	9353	75320
1110011	548	7	2	1	1	2	2	1	2	2	2	1	2	1	2	2	2	2224	358	2	62608	2307	361	1	63042	2446	415	2	64702	2514	404	6	62972
1110011	549	2	2	3	2	2	2	3	2	2	2	3	2	2	2	3	2	2218	309	364	64245	2275	345	360	65047	2478	371	429	66316	2494	379	389	64593
1110100	550	2	1	0	0	1	1	1	0	0	1	1	0	0	0	1	1	0	0	1	0	0	1	0	1	1	1	0	0	0	1	1	1
1110101	551	82	77	63	369	24	32	31	366	23	25	39	399	24	23	78	343	18	17	80	369	17	18	84	392	18	24	32	399	43	18	34	343
1110111	552	2	2	3	257	2	2	3	230	2	2	3	231	2	2	3	258	2	2	3	234	2	3	3	236	2	2	3	232	2	2	3	229
1111001	553	2	2	3	59	2	2	3	49	2	2	3	48	2	2	3	65	10464	8589	8628	75684	10698	8871	8893	77679	11167	9165	9258	76654	11426	9348	9415	76814
1111011	554	2	2	1	1	2	2	1	2	2	2	1	2	2	2	2	1	2186	376	1	62538	2304	370	1	64600	2421	422	2	64723	2504	422	1	62993
1111011	555	2	2	3	2	2	2	3	2	2	2	3	2	2	2	3	2	2216	365	334	64240	2277	344	337	66395	2500	372	364	66326	2491	374	391	64618

continued on next page

Table A.7: *NPMan* absolute (ms): average over all iterations of a query with DB intervals
(continued from previous page)

class	ID	+DB intervals																															
		+indirect keyword occurrences																−indirect keyword occurrences															
		+index intervals								−index intervals								+index intervals								−index intervals							
		+label index				−label index				+label index				−label index				+label index				−label index				+label index				−label index			
		ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG
		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
1111100	556	13	16	19	341	12	12	19	301	13	63	14	308	13	13	19	318	14	14	15	317	15	15	22	296	15	15	16	319	16	15	17	294
1111100	557	3	1	2	16	3	1	1	2	3	1	7	3	3	1	2	2	3	1	2	3	3	1	2	3	3	1	2	3	3	1	2	2
1111101	558	65	54	62	85	55	55	116	111	56	61	70	87	84	58	99	122	61	61	68	95	62	63	74	119	70	65	67	100	66	67	77	104
1111111	559	2	2	1	2	1	1	2	2	2	2	2	2	2	2	1	2	2	2	7	1	2	2	2	2	2	2	1	2	2	2	1	2
1111111	560	7	4	5	10	3	3	9	11	4	4	5	11	3	4	5	10	3	3	5	10	4	4	5	11	4	4	11	10	4	4	5	11

Table A.8: *NPMan* absolute (ms): average over all iterations of a query without DB intervals

class	ID	−DB intervals																															
		+indirect keyword occurrences																−indirect keyword occurrences															
		+index intervals								−index intervals								+index intervals								−index intervals							
		+label index				−label index				+label index				−label index				+label index				−label index				+label index				−label index			
		ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG
		32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
0001111	500	432	171	160	711	484	210	237	1002	274	179	200	711	453	195	227	895	379	166	169	755	607	218	263	965	415	175	204	677	622	211	224	902
0001111	501	431	294	349	1581	595	335	349	1859	373	284	330	1747	598	334	332	1682	537	304	344	1440	774	385	371	1746	542	348	317	1599	773	381	360	1751
0100101	502	3518	4233	4300	13667	3479	4313	4334	14076	3572	4441	4650	14074	3598	4644	4684	14498	5423	4809	4798	14705	5583	4896	4948	15118	5742	5025	5087	14954	5920	5144	5156	15042
0100111	503	178	7	9	2118	223	17	22	2484	168	8	62	2560	251	17	18	2386	271	8	9	2407	312	18	19	2472	244	8	9	2277	309	18	19	2380
0100111	504	593	223	223	9620	664	174	185	9538	672	206	195	9458	658	195	242	9229	842	233	197	9825	907	192	208	9538	916	206	233	9684	964	229	283	9369
0100111	505	220	64	73	6886	298	73	83	6482	294	75	73	6483	335	80	81	6161	331	72	73	6178	352	116	91	7038	331	77	79	6178	366	91	87	6247
0100111	506	1946	371	367	23751	1906	389	375	23495	2018	416	394	23540	2021	424	411	23768	2809	413	396	22459	2871	440	429	23438	3019	440	446	22493	3064	462	525	23040
0100111	507	526	118	152	9815	626	124	203	9357	646	134	144	9222	650	141	144	9455	796	132	165	9091	898	143	174	9517	847	142	151	9158	941	147	179	9833
0101101	508	159	167	182	426	259	200	197	550	212	176	217	459	261	248	233	562	240	190	193	584	314	224	222	586	285	238	229	513	335	242	233	536
0101111	509	55	15	20	294	127	27	34	356	81	15	17	434	126	27	28	382	70	16	17	362	166	33	90	394	81	17	18	451	178	28	30	359
0101111	510	25	14	16	45	141	23	32	101	23	15	17	41	126	24	35	93	53	16	18	46	99	30	41	97	35	17	24	48	86	25	36	95
0101111	511	98	17	23	230	194	30	80	331	56	18	19	287	150	30	44	369	70	18	20	428	151	30	106	388	94	19	21	423	142	31	44	345
0110001	512	2	2	3	34	1	2	3	15	1	2	3	15	2	2	3	15	11416	9154	9145	32935	11626	9314	9378	33402	11945	9502	9598	33314	12174	9674	9753	33178
0110011	513	2	2	3	2	2	2	3	2	1	2	3	2	1	2	3	2	2796	415	419	24014	2905	469	411	23241	3004	453	447	23356	3066	450	432	22872
0110101	514	2820	3616	3747	3880	2931	3697	3709	3968	2901	3759	3946	4112	2916	3857	3931	4140	3902	3960	3996	4208	3990	4099	4103	4312	4075	4077	4155	4416	4156	4202	4212	4465
0110111	515	6	7	12	161	6	7	8	156	10	7	8	157	6	7	8	160	8	8	9	158	7	7	9	159	7	8	9	158	8	7	9	159
0111001	516	7	2	3	15	2	2	3	14	1	1	3	15	2	2	3	15	11390	9129	9177	32802	11633	9319	9410	32625	11981	9541	9558	33289	12183	9644	9759	33282
0111011	517	2	2	3	2	2	2	3	2	2	2	3	2	2	2	3	2	2792	413	393	24331	2853	402	416	22643	3004	436	461	22729	3036	453	502	23043
0111100	518	38	43	67	42	37	51	49	43	59	45	46	45	34	77	67	46	47	74	54	47	48	55	59	48	50	50	57	49	51	51	51	61
0111100	519	24	37	38	100	23	33	35	100	24	35	36	123	24	36	37	158	37	37	38	120	37	48	44	124	39	38	40	149	39	40	41	127
0111101	520	168	153	163	173	128	156	195	182	132	158	164	180	138	165	168	185	193	175	228	207	171	188	179	191	174	175	177	248	176	190	184	195
0111111	521	6	7	9	11	6	8	9	13	6	9	10	13	7	9	9	14	8	8	10	13	9	9	16	12	9	9	10	14	8	9	15	13
0111111	522	5	6	7	256	5	6	7	250	9	6	7	244	5	7	7	248	6	7	7	253	6	6	8	253	7	7	8	229	7	12	8	249
1001111	523	71	20	25	423	167	33	38	432	96	20	27	463	164	34	49	496	93	21	43	601	217	40	120	509	124	22	24	440	188	40	47	471
1001111	524	264	127	150	506	443	165	210	771	289	134	212	557	447	199	191	773	351	144	157	596	599	192	189	810	353	158	187	602	606	209	204	793
1001111	525	354	280	303	1243	597	332	313	1340	378	271	287	1244	587	322	350	1327	518	287	348	1087	753	343	361	1377	531	317	317	1128	760	345	367	1374
1011111	526	9	11	13	362	10	12	14	327	9	12	19	386	9	13	19	358	13	13	14	371	13	19	15	333	13	13	15	339	14	19	15	347
1100100	527	36	24	32	25	110	70	114	59	32	24	35	24	64	59	114	94	29	24	84	24	74	75	69	53	24	30	45	24	57	69	70	50

continued on next page

Table A.8: *NPMan* absolute (ms): average over all iterations of a query without DB intervals
(continued from previous page)

class	ID	−DB intervals																															
		+indirect keyword occurrences																−indirect keyword occurrences															
		+index intervals								−index intervals								+index intervals								−index intervals							
		+label index				−label index				+label index				−label index				+label index				−label index				+label index				−label index			
		ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG
		32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
1100101	528	584	149	241	10690	666	156	172	10373	628	158	179	10158	654	207	213	10289	834	184	179	9707	874	183	252	10473	894	176	190	10707	980	188	219	10071
1100111	529	1856	375	363	23171	1930	378	405	23729	1971	396	417	22816	2011	423	423	22331	2805	379	453	24959	2899	400	482	23717	2994	440	501	23933	3042	463	432	22902
1100111	530	529	115	150	9167	569	121	174	9509	611	125	143	9645	658	138	183	9595	783	152	155	9219	853	151	144	9572	857	137	143	9506	887	151	149	9474
1100111	531	206	85	70	6126	277	75	81	6182	259	71	79	5564	261	75	85	6511	302	68	75	6106	373	136	78	5753	297	73	75	6175	391	85	93	6227
1100111	532	158	3	9	2306	214	20	18	2391	164	4	5	2125	256	13	14	2244	214	4	5	2376	315	13	15	2192	255	4	6	2239	302	13	15	2383
1100111	533	576	127	133	9269	576	154	138	9662	602	147	153	9382	610	146	218	9497	794	141	136	9414	834	184	180	9526	865	161	147	9272	886	156	240	9796
1100111	534	1882	423	2	21883	1868	428	2	21137	2005	428	1	21925	2051	452	1	21884	2809	489	2	22257	2838	537	2	21834	3012	479	2	22837	3070	501	1	21631
1101100	535	3	3	3	3	125	83	154	77	3	3	3	3	80	80	83	69	2	3	3	3	84	88	84	95	3	3	3	3	81	83	84	74
1101101	536	45	14	16	279	148	34	36	348	53	16	17	314	124	53	29	389	67	16	17	296	156	29	32	389	72	17	18	315	138	32	31	383
1101110	537	420	169	485	270	494	416	417	300	464	395	465	254	439	252	433	337	222	421	448	255	337	466	466	313	245	226	421	258	303	252	395	331
1101111	538	50	9	1	258	125	22	7	313	47	10	1	276	127	23	1	348	65	10	1	247	135	23	1	329	70	10	2	393	159	23	2	341
1101111	539	57	10	11	281	123	21	26	316	53	10	12	228	119	21	57	316	65	10	11	217	135	32	23	368	87	11	13	290	135	31	34	344
1101111	540	20	8	2	82	122	21	1	157	21	8	2	85	97	26	1	157	30	9	2	75	131	22	2	151	32	16	1	86	110	29	2	148
1101111	541	2	1	1	2	84	29	1	105	2	0	2	1	104	78	1	85	2	1	1	1	86	23	1	73	2	1	2	2	71	13	1	70
1101111	542	43	12	33	267	123	24	77	344	47	12	14	243	163	24	29	291	65	12	66	247	135	25	26	335	70	13	23	279	141	34	30	325
1101111	543	578	122	1	9200	564	234	2	9107	591	143	2	9179	638	229	1	9418	798	161	1	9279	827	167	2	9591	847	170	2	8924	894	201	1	9315
1101111	544	14	22	1	26	96	30	1	96	15	2	2	27	91	15	2	108	20	2	2	28	97	23	2	113	22	2	2	29	100	69	2	124
1101111	545	2	0	1	2	105	114	2	52	2	0	2	2	61	60	2	56	2	1	2	2	64	65	1	58	2	0	1	2	49	61	2	61
1101111	546	14	3	5	28	79	21	13	89	15	3	5	28	77	16	13	79	20	3	5	30	81	21	17	91	21	4	5	36	80	65	12	81
1110001	547	2	2	3	15	2	9	4	15	2	3	3	15	2	2	3	15	11444	9194	9214	32223	11630	9357	9378	33441	11959	9545	9556	32837	12206	9697	9753	32843
1110011	548	2	2	1	2	2	2	1	2	1	2	2	2	1	2	1	2	2779	480	2	21426	2845	459	1	22954	2956	501	2	23317	3044	518	1	21606
1110011	549	2	2	3	2	2	2	52	2	2	2	9	2	2	2	3	2	2764	425	482	22482	2852	473	414	24531	2962	416	453	24372	3014	453	458	22548
1110100	550	0	0	1	0	1	1	1	0	1	1	1	1	0	6	1	0	0	1	1	0	0	0	1	0	0	0	1	1	1	0	1	1
1110101	551	14	10	11	246	67	20	20	246	17	14	27	222	18	20	21	222	10	15	74	251	10	75	17	245	10	10	11	242	10	71	67	247
1110111	552	2	2	3	153	2	2	3	155	2	2	3	151	2	2	3	154	2	2	3	181	2	2	3	170	2	2	4	150	3	2	4	154
1111001	553	2	2	3	14	2	2	3	15	2	2	3	15	2	52	3	15	11419	9133	9164	32574	11758	9295	9380	34480	11932	9528	9545	34531	12190	9650	9790	33403
1111011	554	2	2	2	2	2	8	1	2	1	2	1	2	2	2	2	2	2741	478	2	21393	2824	495	1	23573	2951	504	2	23373	3018	503	1	22822
1111011	555	2	3	3	2	2	9	3	2	2	2	3	2	2	2	3	2	2791	433	434	22444	2852	436	406	24626	3005	417	453	24421	3057	455	459	26626

continued on next page

Table A.8: *NPMan* absolute (ms): average over all iterations of a query without DB intervals
(continued from previous page)

class	ID	−DB intervals																															
		+indirect keyword occurrences																−indirect keyword occurrences															
		+index intervals								−index intervals								+index intervals								−index intervals							
		+label index				−label index				+label index				−label index				+label index				−label index				+label index				−label index			
		ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG	ID	SG	SL	DG
		32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
1111100	556	7	9	11	107	7	10	11	103	7	10	11	127	7	10	17	95	10	10	12	77	10	11	12	124	10	11	12	77	11	31	62	86
1111100	557	4	1	7	3	3	1	1	3	3	1	7	3	7	2	2	3	3	1	2	3	3	1	2	3	3	1	2	3	3	1	2	4
1111101	558	32	39	93	55	37	46	42	61	32	43	49	57	36	43	49	57	43	49	50	80	44	44	46	60	45	45	67	80	45	46	81	84
1111111	559	2	2	1	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	1	2	2	8	2	2	2	2	2	2	2	2	2	3
1111111	560	3	3	4	7	3	3	4	7	3	3	4	8	3	4	4	8	4	3	5	7	3	4	5	7	3	4	10	7	3	3	5	9

Appendix B

Sample queries from the test suite *CitiesMan*

Table B.1 lists all queries evaluated in the test suite *CitiesMan* (see section 5.2 for an overview of the different test suites and query sets used for experimental evaluation). The queries are given in an alternative syntax, differing from the XPath-based notation used throughout this work (see section 2.3) in that query paths are not nested. Instead the join query nodes are duplicated for all paths attached to them. Every query node is represented by an identifier beginning with a capital letter (S for structural query nodes, C for matching textual query nodes, and M for mismatching textual query nodes). Two nodes may be linked by either a rigid edge (−) or a soft edge (∼). Multiple pairs of linked query node are combined by the + operator to form paths and trees. Unlike the graphical representation used in chapter 2 and 3, labels and query keywords are attached to query nodes instead of edges (which makes no difference in case of tree-shaped data). These label or keyword specifications are also combined with the + operator. Thus S1−S2+transport(S2) denotes a structural query node S2 reached from another structural query node S1 by a rigid edge labelled transport. Similarly, S1−M0+mismatch_keyword#M0 represents a textual query node M0 which is a child of a structural query node S1 and which contains the query keyword *"mismatch_keyword"*. (The keyword operator # must not be confused with the prefix used for index node identifiers.) Unspecified labels (∗ in XPath notation) are simply omitted. The query trees shown in figure 2.1 on page 13 are equivalent to the following two query expressions:

figure 2.1 *(a)*:
S0+book(S0)+S0−S1+preface(S1)+S1−S2+paragraph(S2)+S2−C0+index#C0+S0∼S3+S3−C1+XML#C1

figure 2.1 *(b)*:
S0+book(S0)+S0−S1+preface(S1)+S1−S2+paragraph(S2)+S2−C0+index#C0+S0−S3+S3∼C1+XML#C1

Table B.1: Queries in the *CitiesMan* test suite

class	ID	query
0111111	63	database(S0)+S0−S1+stadt(S1)+S1−S2+gastronomie(S2)+S2−S3+restaurant(S3)+S3−S4+oeffnungszeiten(S4)+S4−C0+vormittag#C0
1111111	m63	database(S0)+S0−S1+stadt(S1)+S1−S2+gastronomie(S2)+S2−S3+restaurant(S3)+S3−S4+oeffnungszeiten(S4)+S4−C0+mismatch_vormittag#C0+S0−M0+mismatch_keyword#M0
0111110	62	database(S0)+S0−S1+stadt(S1)+S1−S2+hotels(S2)+S2−S3+hotel(S3)+S3−S4+vorwahl(S4)+S4−C0+08821#C0
1111110	m62	database(S0)+S0−S1+stadt(S1)+S1−S2+hotels(S2)+S2−S3+hotel(S3)+S3−S4+vorwahl(S4)+S4−C0+mismatch_08821#C0+S0−M0+mismatch_keyword#M0
0111101	61	database(S0)+S0−S1+stadt(S1)+S1−S2+transport(S2)+S2−S3+anreise(S3)+S3−S4+flughafen(S4)+S4−S5+name(S5)+S5−C0+josef#C0
1111101	m61	database(S0)+S0−S1+stadt(S1)+S1−S2+transport(S2)+S2−S3+anreise(S3)+S3−S4+flughafen(S4)+S4−S5+name(S5)+S5−C0+mismatch_josef#C0+S0−M0+mismatch_keyword#M0
0111100	60b	database(S0)+S0−S1+stadt(S1)+S1−S2+freizeitangebote(S2)+S2−S3+buehne(S3)+S3−S4+theater(S4)+S4−S5+beschreibung(S5)+S5−C0+und#C0
1111100	m60b	database(S0)+S0−S1+stadt(S1)+S1−S2+freizeitangebote(S2)+S2−S3+buehne(S3)+S3−S4+theater(S4)+S4−S5+beschreibung(S5)+S5−C0+mismatch_und#C0+S0−M0+mismatch_keyword#M0
0111100	60a	database(S0)+S0−S1+stadt(S1)+S1−S2+freizeitangebote(S2)+S2−S3+buehne(S3)+S3−S4+theater(S4)+S4−S5+beschreibung(S5)
1111100	m60a	database(S0)+S0−S1+stadt(S1)+S1−S2+freizeitangebote(S2)+S2−S3+buehne(S3)+S3−S4+theater(S4)+S4−S5+beschreibung(S5)+S0−M0+mismatch_keyword#M0
0111011	59	database(S0)+S0−S1+stadt(S1)+S1∼C0+vormittag#C0
1111011	m59	database(S0)+S0−S1+stadt(S1)+S1∼C0+mismatch_vormittag#C0+S0−M0+mismatch_keyword#M0
0111010	58	database(S0)+S0−S1+stadt(S1)+S1∼C0+08821#C0
1111010	m58	database(S0)+S0−S1+stadt(S1)+S1∼C0+mismatch_08821#C0+S0−M0+mismatch_keyword#M0
0111001	57	database(S0)+S0−S1+stadt(S1)+S1∼C0+josef#C0
1111001	m57	database(S0)+S0−S1+stadt(S1)+S1∼C0+mismatch_josef#C0+S0−M0+mismatch_keyword#M0
0111000	56	database(S0)+S0−S1+stadt(S1)+S1∼C0+und#C0
1111000	m56	database(S0)+S0−S1+stadt(S1)+S1∼C0+mismatch_und#C0+S0−M0+mismatch_keyword#M0
0110111	55	S0−S1+S1−S2+S2−S3+S3−S4+S4−C0+vormittag#C0
1110111	m55	S0−S1+S1−S2+S2−S3+S3−S4+S4−C0+mismatch_vormittag#C0+S0−M0+mismatch_keyword#M0
0110110	54	S0−S1+S1−S2+S2−S3+S3−S4+S4−C0+08821#C0
1110110	m54	S0−S1+S1−S2+S2−S3+S3−S4+S4−C0+mismatch_08821#C0+S0−M0+mismatch_keyword#M0
0110101	53	S0−S1+S1−S2+S2−S3+S3−S4+S4−S5+S5−C0+josef#C0
1110101	m53	S0−S1+S1−S2+S2−S3+S3−S4+S4−S5+S5−C0+mismatch_josef#C0+S0−M0+mismatch_keyword#M0
0110100	52b	S0−S1+S1−S2+S2−S3+S3−S4+S4−C0+und#C0
1110100	m52b	S0−S1+S1−S2+S2−S3+S3−S4+S4−C0+mismatch_und#C0+S0−M0+mismatch_keyword#M0
0110100	52a	S0−S1+S1−S2+S2−S3+S3−S4
1110100	m52a	S0−S1+S1−S2+S2−S3+S3−S4+S0−M0+mismatch_keyword#M0
0110011	51	S0−S1+S1∼C0+vormittag#C0
1110011	m51	S0−S1+S1∼C0+mismatch_vormittag#C0+S0−M0+mismatch_keyword#M0

continued on next page

Table B.1: Queries in the *CitiesMan* test suite
(continued from previous page)

class	ID	query
0110010	50	S0−S1+S1∼C0+08821#C0
1110010	m50	S0−S1+S1∼C0+mismatch_08821#C0+S0−M0+mismatch_keyword#M0
0110001	49	S0−S1+S1∼C0+josef#C0
1110001	m49	S0−S1+S1∼C0+mismatch_josef#C0+S0−M0+mismatch_keyword#M0
0110000	48	S0−S1+S1∼C0+und#C0
1110000	m48	S0−S1+S1∼C0+mismatch_und#C0+S0−M0+mismatch_keyword#M0
0101111	47	∼S0+restaurant(S0)+S0−S1+oeffnungszeiten(S1)+S1−C0+vormittag#C0
1101111	m47	∼S0+restaurant(S0)+S0−S1+oeffnungszeiten(S1)+S1−C0+mismatch_vormittag#C0+S0−M0+mismatch_keyword#M0
0101110	46	∼S0+hotel(S0)+S0−S1+vorwahl(S1)+S1−C0+08821#C0
1101110	m46	∼S0+hotel(S0)+S0−S1+vorwahl(S1)+S1−C0+mismatch_08821#C0+S0−M0+mismatch_keyword#M0
0101101	45	∼S0+flughafen(S0)+S0−S1+name(S1)+S1−C0+josef#C0
1101101	m45	∼S0+flughafen(S0)+S0−S1+name(S1)+S1−C0+mismatch_josef#C0+S0−M0+mismatch_keyword#M0
0101100	44b	∼S0+theater(S0)+S0−S1+beschreibung(S1)+S1−C0+und#C0
1101100	m44b	∼S0+theater(S0)+S0−S1+beschreibung(S1)+S1−C0+mismatch_und#C0+S0−M0+mismatch_keyword#M0
0101100	44a	∼S0+theater(S0)+S0−S1+beschreibung(S1)
1101100	m44a	∼S0+theater(S0)+S0−S1+beschreibung(S1)+S0−M0+mismatch_keyword#M0
0101011	43	∼S0+gastronomie(S0)+S0∼C0+vormittag#C0
1101011	m43	∼S0+gastronomie(S0)+S0∼C0+mismatch_vormittag#C0+S0−M0+mismatch_keyword#M0
0101010	42	∼S0+hotels(S0)+S0∼C0+08821#C0
1101010	m42	∼S0+hotels(S0)+S0∼C0+mismatch_08821#C0+S0−M0+mismatch_keyword#M0
0101001	41	∼S0+transport(S0)+S0∼C0+josef#C0
1101001	m41	∼S0+transport(S0)+S0∼C0+mismatch_josef#C0+S0−M0+mismatch_keyword#M0
0101000	40	∼S0+buehne(S0)+S0∼C0+und#C0
1101000	m40	∼S0+buehne(S0)+S0∼C0+mismatch_und#C0+S0−M0+mismatch_keyword#M0
0100111	39b	∼S0+adresse(S0)+S0−S1+strasse_nummer(S1)+S1−C0+graseck#C0
1100111	m39b	∼S0+adresse(S0)+S0−S1+strasse_nummer(S1)+S1−C0+mismatch_graseck#C0+S0−M0+mismatch_keyword#M0
0100111	39a	∼S0+S0−C0+vormittag#C0
1100111	m39a	∼S0+S0−C0+mismatch_vormittag#C0+S0−M0+mismatch_keyword#M0
0100110	38b	∼S0+vorwahl(S0)+S0−C0+08821#C0
1100110	m38b	∼S0+vorwahl(S0)+S0−C0+mismatch_08821#C0+S0−M0+mismatch_keyword#M0

continued on next page

Table B.1: Queries in the *CitiesMan* test suite
(continued from previous page)

class	ID	query
0100110	38a	∼S0+S0−C0+08821#C0
1100110	m38a	∼S0+S0−C0+mismatch_08821#C0+S0−M0+mismatch_keyword#M0
0100101	37b	∼S0+name(S0)+S0−C0+josef#C0
1100101	m37b	∼S0+name(S0)+S0−C0+mismatch_josef#C0+S0−M0+mismatch_keyword#M0
0100101	37a	∼S0+S0−C0+josef#C0
1100101	m37a	∼S0+S0−C0+mismatch_josef#C0+S0−M0+mismatch_keyword#M0
0100100	36d	∼S0+adresse(S0)+S0−S1+strasse_nummer(S1)+S1−C0+4#C0
1100100	m36d	∼S0+adresse(S0)+S0−S1+strasse_nummer(S1)+S1−C0+mismatch_4#C0+S0−M0+mismatch_keyword#M0
0100100	36c	∼S0+adresse(S0)
1100100	m36c	∼S0+adresse(S0)+S0−M0+mismatch_keyword#M0
0100100	36b	∼S0+S0−C0+und#C0
1100100	m36b	∼S0+S0−C0+mismatch_und#C0+S0−M0+mismatch_keyword#M0
0100100	36a	∼S0
1100100	m36a	∼S0+S0−M0+mismatch_keyword#M0
0100011	35b	∼S0+adresse(S0)+S0∼C0+graseck#C0
1100011	m35b	∼S0+adresse(S0)+S0∼C0+mismatch_graseck#C0+S0−M0+mismatch_keyword#M0
0100011	35a	∼S0+S0∼C0+vormittag#C0
1100011	m35a	∼S0+S0∼C0+mismatch_vormittag#C0+S0−M0+mismatch_keyword#M0
0100010	34b	∼S0+vorwahl(S0)+S0∼C0+08821#C0
1100010	m34b	∼S0+vorwahl(S0)+S0∼C0+mismatch_08821#C0+S0−M0+mismatch_keyword#M0
0100010	34a	∼S0+S0∼C0+08821#C0
1100010	m34a	∼S0+S0∼C0+mismatch_08821#C0+S0−M0+mismatch_keyword#M0
0100001	33b	∼S0+name(S0)+S0∼C0+josef#C0
1100001	m33b	∼S0+name(S0)+S0∼C0+mismatch_josef#C0+S0−M0+mismatch_keyword#M0
0100001	33a	∼S0+S0∼C0+josef#C0
1100001	m33a	∼S0+S0∼C0+mismatch_josef#C0+S0−M0+mismatch_keyword#M0
0100000	32b	∼S0+adresse(S0)+S0∼C0+4#C0
1100000	m32b	∼S0+adresse(S0)+S0∼C0+mismatch_4#C0+S0−M0+mismatch_keyword#M0
0100000	32a	∼S0+S0∼C0+und#C0
1100000	m32a	∼S0+S0∼C0+mismatch_und#C0+S0−M0+mismatch_keyword#M0

continued on next page

Table B.1: Queries in the *CitiesMan* test suite
(continued from previous page)

class	ID	query
0011111	31	database(S0)+S0−S1+stadt(S1)+S1−S2+gastronomie(S2)+S2−S3+restaurant(S3)+S3−S4+name(S4)+S4−C0+riessersee#C0+S3−S5+adresse(S5)+S5−S6+strasse_nummer(S6)+S6−C1+riess#C1+S1−S7+hotels(S7)+S7−S8+hotel(S8)+S8−S9+adresse(S9)+S9−S10+strasse_nummer(S10)+S10−C2+badgasse#C2+S1−S11+freizeitangebote(S11)+S11−S12+sport(S12)+S12−S13+sportangebot(S13)+S13−S14+adresse(S14)+S14−S15+strasse_nummer(S15)+S15−C3+vordergraseck#C3+S13−S16+name(S16)+S16−C4+kegelbahnen#C4
1011111	m31	database(S0)+S0−S1+stadt(S1)+S1−S2+gastronomie(S2)+S2−S3+restaurant(S3)+S3−S4+name(S4)+S4−C0+mismatch_riessersee#C0+S3−S5+adresse(S5)+S5−S6+strasse_nummer(S6)+S6−C1+mismatch_riess#C1+S1−S7+hotels(S7)+S7−S8+hotel(S8)+S8−S9+adresse(S9)+S9−S10+strasse_nummer(S10)+S10−C2+mismatch_badgasse#C2+S1−S11+freizeitangebote(S11)+S11−S12+sport(S12)+S12−S13+sportangebot(S13)+S13−S14+adresse(S14)+S14−S15+strasse_nummer(S15)+S15−C3+mismatch_vordergraseck#C3+S13−S16+name(S16)+S16−C4+mismatch_kegelbahnen#C4+S0−M0+mismatch_keyword#M0
0011110	30	database(S0)+S0−S1+stadt(S1)+S1−S2+hotels(S2)+S2−S3+hotel(S3)+S3−S4+vorwahl(S4)+S4−C0+08821#C0+S1−S5+freizeitangebote(S5)+S5−S6+sport(S6)+S6−S7+sportangebot(S7)+S7−S8+adresse(S8)+S8−S9+plz(S9)+S9−C1+82467#C1+S0−S10+stadt(S10)+S10−S11+freizeitangebote(S11)+S11−S12+veranstaltungen(S12)+S12−S13+veranstaltung(S13)+S13−S14+vorwahl(S14)+S14−C2+08031#C2
1011110	m30	database(S0)+S0−S1+stadt(S1)+S1−S2+hotels(S2)+S2−S3+hotel(S3)+S3−S4+vorwahl(S4)+S4−C0+mismatch_08821#C0+S1−S5+freizeitangebote(S5)+S5−S6+sport(S6)+S6−S7+sportangebot(S7)+S7−S8+adresse(S8)+S8−S9+plz(S9)+S9−C1+mismatch_82467#C1+S0−S10+stadt(S10)+S10−S11+freizeitangebote(S11)+S11−S12+veranstaltungen(S12)+S12−S13+veranstaltung(S13)+S13−S14+vorwahl(S14)+S14−C2+mismatch_08031#C2+S0−M0+mismatch_keyword#M0
0011101	29	database(S0)+S0−S1+stadt(S1)+S1−S2+transport(S2)+S2−S3+anreise(S3)+S3−S4+flughafen(S4)+S4−S5+name(S5)+S5−C0+josef#C0+S1−S6+hotels(S6)+S6−S7+hotel(S7)+S7−S8+bettenanzahl(S8)+S8−C1+17#C1+S6−S9+hotel(S9)+S9−S10+anfahrt(S10)+S10−C2+23#C2+S6−S11+hotel(S11)+S11−S12+beschreibung(S12)+S12−C3+35#C3+S6−S13+hotel(S13)+S13−S14+beschreibung(S14)+S14−C4+70#C4
1011101	m29	database(S0)+S0−S1+stadt(S1)+S1−S2+transport(S2)+S2−S3+anreise(S3)+S3−S4+flughafen(S4)+S4−S5+name(S5)+S5−C0+mismatch_josef#C0+S1−S6+hotels(S6)+S6−S7+hotel(S7)+S7−S8+bettenanzahl(S8)+S8−C1+mismatch_17#C1+S6−S9+hotel(S9)+S9−S10+anfahrt(S10)+S10−C2+mismatch_23#C2+S6−S11+hotel(S11)+S11−S12+beschreibung(S12)+S12−C3+mismatch_35#C3+S6−S13+hotel(S13)+S13−S14+beschreibung(S14)+S14−C4+mismatch_70#C4+S0−M0+mismatch_keyword#M0
0011100	28	database(S0)+S0−S1+stadt(S1)+S1−S2+gastronomie(S2)+S2−S3+restaurant(S3)+S3−S4+name(S4)+S4−C0+und#C0+S2−S5+restaurant(S5)+S5−S6+adresse(S6)+S6−S7+strasse_nummer(S7)+S7−C2+der#C2+S1−S8+hotels(S8)+S8−S9+hotel(S9)+S9−S10+adresse(S10)+S10−S11+strasse_nummer(S11)+S11−C1+am#C1+S1−S12+freizeitangebote(S12)+S12−S13+sport(S13)+S13−S14+sportangebot(S14)+S14−S15+adresse(S15)+S15−S16+ort(S16)+S16−C3+augsburg#C3+S13−S17+sportangebot(S17)+S17−S18+beschreibung(S18)+S18−C4+die#C4
1011100	m28	database(S0)+S0−S1+stadt(S1)+S1−S2+gastronomie(S2)+S2−S3+restaurant(S3)+S3−S4+name(S4)+S4−C0+mismatch_und#C0+S2−S5+restaurant(S5)+S5−S6+adresse(S6)+S6−S7+strasse_nummer(S7)+S7−C2+mismatch_der#C2+S1−S8+hotels(S8)+S8−S9+hotel(S9)+S9−S10+adresse(S10)+S10−S11+strasse_nummer(S11)+S11−C1+mismatch_am#C1+S1−S12+freizeitangebote(S12)+S12−S13+sport(S13)+S13−S14+sportangebot(S14)+S14−S15+adresse(S15)+S15−S16+ort(S16)+S16−C3+mismatch_augsburg#C3+S13−S17+sportangebot(S17)+S17−S18+beschreibung(S18)+S18−C4+mismatch_die#C4+S0−M0+mismatch_keyword#M0
0011011	27	database(S0)+S0−S1+stadt(S1)+S1∼C0+riessersee#C0+S1∼C1+badgasse#C1+S1∼C2+riess#C2+S1∼C3+vordergraseck#C3+S1∼C4+kegelbahnen#C4
1011011	m27	database(S0)+S0−S1+stadt(S1)+S1∼C0+mismatch_riessersee#C0+S1∼C1+mismatch_badgasse#C1+S1∼C2+mismatch_riess#C2+S1∼C3+mismatch_vordergraseck#C3+S1∼C4+mismatch_kegelbahnen#C4+S0−M0+mismatch_keyword#M0
0011010	26	database(S0)+S0−S1+stadt(S1)+S1∼C0+08821#C0+S1∼C1+82467#C1+S0−S2+stadt(S2)+S2∼C2+08031#C2
1011010	m26	database(S0)+S0−S1+stadt(S1)+S1∼C0+mismatch_08821#C0+S1∼C1+mismatch_82467#C1+S0−S2+stadt(S2)+S2∼C2+mismatch_08031#C2+S0−M0+mismatch_keyword#M0
0011001	25	database(S0)+S0−S1+stadt(S1)+S1∼C0+josef#C0+S1∼C1+17#C1+S1∼C2+23#C2+S1∼C3+35#C3+S1∼C4+70#C4
1011001	m25	database(S0)+S0−S1+stadt(S1)+S1∼C0+mismatch_josef#C0+S1∼C1+mismatch_17#C1+S1∼C2+mismatch_23#C2+S1∼C3+mismatch_35#C3+S1∼C4+mismatch_70#C4+S0−M0+mismatch_keyword#M0
0011000	24	database(S0)+S0−S1+stadt(S1)+S1∼C0+und#C0+S1∼C1+am#C1+S1∼C2+der#C2+S1∼C3+augsburg#C3+S1∼C4+die#C4
1011000	m24	database(S0)+S0−S1+stadt(S1)+S1∼C0+mismatch_und#C0+S1∼C1+mismatch_am#C1+S1∼C2+mismatch_der#C2+S1∼C3+mismatch_augsburg#C3+S1∼C4+mismatch_die#C4+S0−M0+mismatch_keyword#M0
0010111	23	S0−S1+S1−S2+S2−S3+S3−S4+S4−C0+riessersee#C0+S3−S5+S5−S6+S6−C1+riess#C1+S1−S7+S7−S8+S8−S9+S9−S10+S10−C2+badgasse#C2+S1−S11+S11−S12+S12−S13+S13−S14+S14−S15+S15−C3+vordergraseck#C3+S13−S16+S16−C4+kegelbahnen#C4
1010111	m23	S0−S1+S1−S2+S2−S3+S3−S4+S4−C0+mismatch_riessersee#C0+S3−S5+S5−S6+S6−C1+mismatch_riess#C1+S1−S7+S7−S8+S8−S9+S9−S10+S10−C2+mismatch_badgasse#C2+S1−S11+S11−S12+S12−S13+S13−S14+S14−S15+S15−C3+mismatch_vordergraseck#C3+S13−S16+S16−C4+mismatch_kegelbahnen#C4+S0−M0+mismatch_keyword#M0
0010110	22	S0−S1+S1−S2+S2−S3+S3−S4+S4−C0+08821#C0+S1−S5+S5−S6+S6−S7+S7−S8+S8−S9+S9−C1+82467#C1+S0−S10+S10−S11+S11−S12+S12−S13+S13−S14+S14−C2+08031#C2
1010110	m22	S0−S1+S1−S2+S2−S3+S3−S4+S4−C0+mismatch_08821#C0+S1−S5+S5−S6+S6−S7+S7−S8+S8−S9+S9−C1+mismatch_82467#C1+S0−S10+S10−S11+S11−S12+S12−S13+S13−S14+S14−C2+mismatch_08031#C2+S0−M0+mismatch_keyword#M0
0010101	21	S0−S1+S1−S2+S2−S3+S3−S4+S4−S5+S5−C0+josef#C0+S1−S6+S6−S7+S7−S8+S8−C1+17#C1+S6−S9+S9−S10+S10−C2+23#C2+S6−S11+S11−S12+S12−C3+35#C3+S6−S13+S13−S14+S14−C4+70#C4
1010101	m21	S0−S1+S1−S2+S2−S3+S3−S4+S4−S5+S5−C0+mismatch_josef#C0+S1−S6+S6−S7+S7−S8+S8−C1+mismatch_17#C1+S6−S9+S9−S10+S10−C2+mismatch_23#C2+S6−S11+S11−S12+S12−C3+mismatch_35#C3+S6−S13+S13−S14+S14−C4+mismatch_70#C4+S0−M0+mismatch_keyword#M0
0010100	20	S0−S1+S1−S2+S2−S3+S3−S4+S4−C0+und#C0+S2−S5+S5−S6+S6−S7+S7−C2+der#C2+S1−S8+S8−S9+S9−S10+S10−S11+S11−C1+am#C1+S1−S12+S12−S13+S13−S14+S14−S15+S15−S16+S16−C3+augsburg#C3+S13−S17+S17−S18+S18−C4+die#C4
1010100	m20	S0−S1+S1−S2+S2−S3+S3−S4+S4−C0+mismatch_und#C0+S2−S5+S5−S6+S6−S7+S7−C2+mismatch_der#C2+S1−S8+S8−S9+S9−S10+S10−S11+S11−C1+mismatch_am#C1+S1−S12+S12−S13+S13−S14+S14−S15+S15−S16+S16−C3+mismatch_augsburg#C3+S13−S17+S17−S18+S18−C4+mismatch_die#C4+S0−M0+mismatch_keyword#M0
0010011	19	S0−S1+S1∼C0+riessersee#C0+S1∼C1+badgasse#C1+S1∼C2+riess#C2+S1∼C3+vordergraseck#C3+S1∼C4+kegelbahnen#C4
1010011	m19	S0−S1+S1∼C0+mismatch_riessersee#C0+S1∼C1+mismatch_badgasse#C1+S1∼C2+mismatch_riess#C2+S1∼C3+mismatch_vordergraseck#C3+S1∼C4+mismatch_kegelbahnen#C4+S0−M0+mismatch_keyword#M0
0010010	18	S0−S1+S1∼C0+08821#C0+S1∼C1+82467#C1+S0−S2+S2∼C2+08031#C2

continued on next page

Table B.1: Queries in the *CitiesMan* test suite
(continued from previous page)

class	ID	query
1010010	m18	S0-S1+S1~C0+mismatch_08821#C0+S1~C1+mismatch_82467#C1+S0-S2+S2~C2+mismatch_08031#C2+S0-M0+mismatch_keyword#M0
0010001	17	S0-S1+S1~C0+josef#C0+S1~C1+17#C1+S1~C2+23#C2+S1~C3+35#C3+S1~C4+70#C4
1010001	m17	S0-S1+S1~C0+mismatch_josef#C0+S1~C1+mismatch_17#C1+S1~C2+mismatch_23#C2+S1~C3+mismatch_35#C3+S1~C4+mismatch_70#C4+S0-M0+mismatch_keyword#M0
0010000	16	S0-S1+S1~C0+und#C0+S1~C1+am#C1+S1~C2+der#C2+S1~C3+augsburg#C3+S1~C4+die#C4
1010000	m16	S0-S1+S1~C0+mismatch_und#C0+S1~C1+mismatch_am#C1+S1~C2+mismatch_der#C2+S1~C3+mismatch_augsburg#C3+S1~C4+mismatch_die#C4+S0-M0+mismatch_keyword#M0
0001111	15	database(S0)+S0~S1+restaurant(S1)+S1-S2+name(S2)+S2-C0+riessersee#C0+S1~S3+adresse(S3)+S3-S4+strasse_nummer(S4)+S4-C1+riess#C1+S0~S5+hotel(S5)+S5-S6+adresse(S6)+S6-S7+strasse_nummer(S7)+S7-C2+badgasse#C2+S0~S8+sportangebot(S8)+S8-S9+adresse(S9)+ S9-S10+strasse_nummer(S10)+S10-C3+vordergraseck#C3+S8-S11+name(S11)+S11-C4+kegelbahnen#C4
1001111	m15	database(S0)+S0~S1+restaurant(S1)+S1-S2+name(S2)+S2-C0+mismatch_riessersee#C0+S1~S3+adresse(S3)+S3-S4+strasse_nummer(S4)+S4-C1+mismatch_riess#C1+S0~S5+hotel(S5)+S5-S6+adresse(S6)+S6-S7+strasse_nummer(S7)+S7-C2+mismatch_badgasse#C2+S0~S8+sportangebot(S8)+ S8-S9+adresse(S9)+S9-S10+strasse_nummer(S10)+S10-C3+mismatch_vordergraseck#C3+S8-S11+name(S11)+S11-C4+mismatch_kegelbahnen#C4+S0-M0+mismatch_keyword#M0
0001110	14	database(S0)+S0~S1+hotel(S1)+S1-S2+vorwahl(S2)+S2~C0+08821#C0+S0~S3+sportangebot(S3)+S3-S4+adresse(S4)+S4-S5+plz(S5)+S5~C1+82467#C1+S0~S6+veranstaltung(S6)+S6-S7+vorwahl(S7)+S7~C2+08031#C2
1001110	m14	database(S0)+S0~S1+hotel(S1)+S1-S2+vorwahl(S2)+S2~C0+mismatch_08821#C0+S0~S3+sportangebot(S3)+S3-S4+adresse(S4)+S4-S5+plz(S5)+S5~C1+mismatch_82467#C1+S0~S6+veranstaltung(S6)+S6-S7+vorwahl(S7)+S7~C2+mismatch_08031#C2+S0-M0+mismatch_keyword#M0
0001101	13	database(S0)+S0~S1+flughafen(S1)+S1-S2+name(S2)+S2-C0+josef#C0+S0~S3+bettenanzahl(S3)+S3-C1+17#C1+S0~S4+hotel(S4)+S4-S5+anfahrt(S5)+S5-C2+23#C2+S0~S6+hotel(S6)+S6-S7+beschreibung(S7)+S7-C3+35#C3+S0~S8+hotel(S8)+S8-S9+beschreibung(S9)+S9-C4+ 70#C4
1001101	m13	database(S0)+S0~S1+flughafen(S1)+S1-S2+name(S2)+S2-C0+mismatch_josef#C0+S0~S3+bettenanzahl(S3)+S3-C1+mismatch_17#C1+S0~S4+hotel(S4)+S4-S5+anfahrt(S5)+S5-C2+mismatch_23#C2+S0~S6+hotel(S6)+S6-S7+beschreibung(S7)+S7-C3+mismatch_35#C3+S0~S8+hotel(S8)+ S8-S9+beschreibung(S9)+S9-C4+mismatch_70#C4+S0-M0+mismatch_keyword#M0
0001100	12	database(S0)+S0~S1+restaurant(S1)+S1-S2+name(S2)+S2-C0+und#C0+S0~S3+hotel(S3)+S3-S4+adresse(S4)+S4-S5+strasse_nummer(S5)+S5-C1+am#C1+S0~S6+restaurant(S6)+S6-S7+adresse(S7)+S7-S8+strasse_nummer(S8)+S8-C2+der#C2+S0~S9+sportangebot(S9)+S9-S10+ adresse(S10)+S10-S11+ort(S11)+S11-C3+augsburg#C3+S0~S12+sportangebot(S12)+S12-S13+beschreibung(S13)+S13-C4+die#C4
1001100	m12	database(S0)+S0~S1+restaurant(S1)+S1-S2+name(S2)+S2-C0+mismatch_und#C0+S0~S3+hotel(S3)+S3-S4+adresse(S4)+S4-S5+strasse_nummer(S5)+S5-C1+mismatch_am#C1+S0~S6+restaurant(S6)+S6-S7+adresse(S7)+S7-S8+strasse_nummer(S8)+S8-C2+mismatch_der#C2+S0~S9+ sportangebot(S9)+S9-S10+adresse(S10)+S10-S11+ort(S11)+S11-C3+mismatch_augsburg#C3+S0~S12+sportangebot(S12)+S12-S13+beschreibung(S13)+S13-C4+mismatch_die#C4+S0-M0+mismatch_keyword#M0
0001011	11	database(S0)+S0~S1+restaurant(S1)+S1~C0+riessersee#C0+S1~C1+riess#C1+S0~S2+hotel(S2)+S2~C2+badgasse#C2+S0~S3+sportangebot(S3)+S3~C3+vordergraseck#C3+S3~C4+kegelbahnen#C4
1001011	m11	database(S0)+S0~S1+restaurant(S1)+S1~C0+mismatch_riessersee#C0+S1~C1+mismatch_riess#C1+S0~S2+hotel(S2)+S2~C2+mismatch_badgasse#C2+S0~S3+sportangebot(S3)+S3~C3+mismatch_vordergraseck#C3+S3~C4+mismatch_kegelbahnen#C4+S0-M0+mismatch_keyword#M0
0001010	10	database(S0)+S0~S1+hotel(S1)+S1~C0+08821#C0+S0~S2+sportangebot(S2)+S2~C1+82467#C1+S0~S3+veranstaltung(S3)+S3~C2+08031#C2
1001010	m10	database(S0)+S0~S1+hotel(S1)+S1~C0+mismatch_08821#C0+S0~S2+sportangebot(S2)+S2~C1+mismatch_82467#C1+S0~S3+veranstaltung(S3)+S3~C2+mismatch_08031#C2+S0-M0+mismatch_keyword#M0
0001001	9	database(S0)+S0~S1+flughafen(S1)+S1~C0+josef#C0+S0~S2+bettenanzahl(S2)+S2~C1+17#C1+S0~S3+hotel(S3)+S3~C2+23#C2+S0~S4+hotel(S4)+S4~C3+35#C3+S0~S5+hotel(S5)+S5~C4+70#C4
1001001	m9	database(S0)+S0~S1+flughafen(S1)+S1~C0+mismatch_josef#C0+S0~S2+bettenanzahl(S2)+S2~C1+mismatch_17#C1+S0~S3+hotel(S3)+S3~C2+mismatch_23#C2+S0~S4+hotel(S4)+S4~C3+mismatch_35#C3+S0~S5+hotel(S5)+S5~C4+mismatch_70#C4+S0-M0+mismatch_keyword#M0
0001000	8	database(S0)+S0~S1+restaurant(S1)+S1~C0+und#C0+S0~S2+hotel(S2)+S2~C1+am#C1+S0~S3+restaurant(S3)+S3~C2+der#C2+S0~S4+sportangebot(S4)+S4~C3+augsburg#C3+S0~S5+sportangebot(S5)+S5~C4+die#C4
1001000	m8	database(S0)+S0~S1+restaurant(S1)+S1~C0+mismatch_und#C0+S0~S2+hotel(S2)+S2~C1+mismatch_am#C1+S0~S3+restaurant(S3)+S3~C2+mismatch_der#C2+S0~S4+sportangebot(S4)+S4~C3+mismatch_augsburg#C3+S0~S5+sportangebot(S5)+S5~C4+mismatch_die#C4+S0-M0+ mismatch_keyword#M0
0000111	7b	S0~S1+name(S1)+S1-C0+riessersee#C0+S0~S2+strasse_nummer(S2)+S2-C1+badgasse#C1+S0~S3+strasse_nummer(S3)+S3-C2+riess#C2+S0~S4+strasse_nummer(S4)+S4-C3+vordergraseck#C3+S0~S5+name(S5)+S5-C4+kegelbahnen#C4
1000111	m7b	S0~S1+name(S1)+S1-C0+mismatch_riessersee#C0+S0~S2+strasse_nummer(S2)+S2-C1+mismatch_badgasse#C1+S0~S3+strasse_nummer(S3)+S3-C2+mismatch_riess#C2+S0~S4+strasse_nummer(S4)+S4-C3+mismatch_vordergraseck#C3+S0~S5+name(S5)+S5-C4+mismatch_kegelbahnen#C4+ S0-M0+mismatch_keyword#M0
0000111	7a	S0~S1+S1-C0+riessersee#C0+S0~S2+S2-C1+badgasse#C1+S0~S3+S3-C2+riess#C2+S0~S4+S4-C3+vordergraseck#C3+S0~S5+S5-C4+kegelbahnen#C4
1000111	m7a	S0~S1+S1-C0+mismatch_riessersee#C0+S0~S2+S2-C1+mismatch_badgasse#C1+S0~S3+S3-C2+mismatch_riess#C2+S0~S4+S4-C3+mismatch_vordergraseck#C3+S0~S5+S5-C4+mismatch_kegelbahnen#C4+S0-M0+mismatch_keyword#M0
0000110	6b	S0~S1+vorwahl(S1)+S1-C0+08821#C0+S0~S2+plz(S2)+S2-C1+82467#C1+S0~S3+vorwahl(S3)+S3-C2+08031#C2
1000110	m6b	S0~S1+vorwahl(S1)+S1-C0+mismatch_08821#C0+S0~S2+plz(S2)+S2-C1+mismatch_82467#C1+S0~S3+vorwahl(S3)+S3-C2+mismatch_08031#C2+S0-M0+mismatch_keyword#M0
0000110	6a	S0~S1+S1-C0+08821#C0+S0~S2+S2-C1+82467#C1+S0~S3+S3-C2+08031#C2
1000110	m6a	S0~S1+S1-C0+mismatch_08821#C0+S0~S2+S2-C1+mismatch_82467#C1+S0~S3+S3-C2+mismatch_08031#C2+S0-M0+mismatch_keyword#M0

continued on next page

Table B.1: Queries in the *CitiesMan* test suite
(continued from previous page)

class	ID	query
0000101	5b	S0∼S1+name(S1)+S1−C0+josef#C0+S0∼S2+strasse_nummer(S2)+S2−C1+17#C1+S0∼S3+strasse_nummer(S3)+S3−C2+23#C2+S0∼S4+beschreibung(S4)+S4−C3+35#C3+S0∼S5+beschreibung(S5)+S5−C4+70#C4
1000101	m5b	S0∼S1+name(S1)+S1−C0+mismatch_josef#C0+S0∼S2+strasse_nummer(S2)+S2−C1+mismatch_17#C1+S0∼S3+strasse_nummer(S3)+S3−C2+mismatch_23#C2+S0∼S4+beschreibung(S4)+S4−C3+mismatch_35#C3+S0∼S5+beschreibung(S5)+S5−C4+mismatch_70#C4+S0−M0+mismatch_keyword#M0
0000101	5a	S0∼S1+S1−C0+josef#C0+S0∼S2+S2−C1+17#C1+S0∼S3+S3−C2+23#C2+S0∼S4+S4−C3+35#C3+S0∼S5+S5−C4+70#C4
1000101	m5a	S0∼S1+S1−C0+mismatch_josef#C0+S0∼S2+S2−C1+mismatch_17#C1+S0∼S3+S3−C2+mismatch_23#C2+S0∼S4+S4−C3+mismatch_35#C3+S0∼S5+S5−C4+mismatch_70#C4+S0−M0+mismatch_keyword#M0
0000100	4b	S0∼S1+name(S1)+S1−C0+und#C0+S0∼S2+strasse_nummer(S2)+S2−C1+am#C1+S0∼S3+strasse_nummer(S3)+S3−C2+der#C2+S0∼S4+ort(S4)+S4−C3+augsburg#C3+S0∼S5+beschreibung(S5)+S5−C4+die#C4
1000100	m4b	S0∼S1+name(S1)+S1−C0+mismatch_und#C0+S0∼S2+strasse_nummer(S2)+S2−C1+mismatch_am#C1+S0∼S3+strasse_nummer(S3)+S3−C2+mismatch_der#C2+S0∼S4+ort(S4)+S4−C3+mismatch_augsburg#C3+S0∼S5+beschreibung(S5)+S5−C4+mismatch_die#C4+S0−M0+mismatch_keyword#M0
0000100	4a	S0∼S1+S1−C0+und#C0+S0∼S2+S2−C1+am#C1+S0∼S3+S3−C2+der#C2+S0∼S4+S4−C3+augsburg#C3+S0∼S5+S5−C4+die#C4
1000100	m4a	S0∼S1+S1−C0+mismatch_und#C0+S0∼S2+S2−C1+mismatch_am#C1+S0∼S3+S3−C2+mismatch_der#C2+S0∼S4+S4−C3+mismatch_augsburg#C3+S0∼S5+S5−C4+mismatch_die#C4+S0−M0+mismatch_keyword#M0
0000011	3b	S0∼S1+S1∼C0+riessersee#C0+S0∼S2+adresse(S2)+S2∼C1+badgasse#C1+S0∼S3+adresse(S3)+S3∼C2+riess#C2+S0∼S4+adresse(S4)+S4∼C3+vordergraseck#C3+S0∼S5+S5∼C4+kegelbahnen#C4
1000011	m3b	S0∼S1+S1∼C0+mismatch_riessersee#C0+S0∼S2+adresse(S2)+S2∼C1+mismatch_badgasse#C1+S0∼S3+adresse(S3)+S3∼C2+mismatch_riess#C2+S0∼S4+adresse(S4)+S4∼C3+mismatch_vordergraseck#C3+S0∼S5+S5∼C4+mismatch_kegelbahnen#C4+S0−M0+mismatch_keyword#M0
0000011	3a	S0∼S1+S1∼C0+riessersee#C0+S0∼S2+S2∼C1+badgasse#C1+S0∼S3+S3∼C2+riess#C2+S0∼S4+S4∼C3+vordergraseck#C3+S0∼S5+S5∼C4+kegelbahnen#C4
1000011	m3a	S0∼S1+S1∼C0+mismatch_riessersee#C0+S0∼S2+S2∼C1+mismatch_badgasse#C1+S0∼S3+S3∼C2+mismatch_riess#C2+S0∼S4+S4∼C3+mismatch_vordergraseck#C3+S0∼S5+S5∼C4+mismatch_kegelbahnen#C4+S0−M0+mismatch_keyword#M0
0000010	2	S0∼S1+S1∼C0+08821#C0+S0∼S2+S2∼C1+82467#C1+S0∼S3+S3∼C2+08031#C2
1000010	m2	S0∼S1+S1∼C0+mismatch_08821#C0+S0∼S2+S2∼C1+mismatch_82467#C1+S0∼S3+S3∼C2+mismatch_08031#C2+S0−M0+mismatch_keyword#M0
0000001	1	S0∼S1+S1∼C0+josef#C0+S0∼S2+S2∼C1+17#C1+S0∼S3+S3∼C2+23#C2+S0∼S4+S4∼C3+35#C3+S0∼S5+S5∼C4+70#C4
1000001	m1	S0∼S1+S1∼C0+mismatch_josef#C0+S0∼S2+S2∼C1+mismatch_17#C1+S0∼S3+S3∼C2+mismatch_23#C2+S0∼S4+S4∼C3+mismatch_35#C3+S0∼S5+S5∼C4+mismatch_70#C4+S0−M0+mismatch_keyword#M0
0000000	0b	S0∼S1+S1∼C0+und#C0+S0∼S2+adresse(S2)+S2∼C1+am#C1+S0∼S3+adresse(S3)+S3∼C2+der#C2+S0∼S4+adresse(S4)+S4∼C3+augsburg#C3+S0∼S5+S5∼C4+die#C4
1000000	m0b	S0∼S1+S1∼C0+mismatch_und#C0+S0∼S2+adresse(S2)+S2∼C1+mismatch_am#C1+S0∼S3+adresse(S3)+S3∼C2+mismatch_der#C2+S0∼S4+adresse(S4)+S4∼C3+mismatch_augsburg#C3+S0∼S5+S5∼C4+mismatch_die#C4+S0−M0+mismatch_keyword#M0
0000000	0a	S0∼S1+S1∼C0+und#C0+S0∼S2+S2∼C1+am#C1+S0∼S3+S3∼C2+der#C2+S0∼S4+S4∼C3+augsburg#C3+S0∼S5+S5∼C4+die#C4
1000000	m0a	S0∼S1+S1∼C0+mismatch_und#C0+S0∼S2+S2∼C1+mismatch_am#C1+S0∼S3+S3∼C2+mismatch_der#C2+S0∼S4+S4∼C3+mismatch_augsburg#C3+S0∼S5+S5∼C4+mismatch_die#C4+S0−M0+mismatch_keyword#M0

Appendix C

Command line interface `caa.db.DBInterpreter`

The following listing shows the command line help of the newly implemented retrieval system interface `caa.db.DBInterpreter` (see section 4.2.1). As can be seen in the first line after the header, the prompt `DBS %>` indicates that the interpreter is ready to accept any of the commands listed below. To display the help, the command `help` is executed. The prompt in the last line shows that after printing the help message, the interpreter again waits for new input to be processed. In this case the session is terminated by the `quit` command.

The help message is divided into three parts. First, a synopsis of different ways to invoke the interpreter is given (section `USAGE`). The next part lists all commands supported by the interpreter (section `INTERPRETER COMMANDS`). Parameters to individual commands are surrounded by angle brackets `<>`. Their respective value ranges are specified in the last part, which makes up the second half of the listing.

```
========================================
This is caa.db.DBInterpreter, $Revision: 1.2 $.
Last modification on $Date: 2003/04/03 18:19:53 $ by $Author: weigel $.
For a list of interpreter commands, please type "help".
Started Tue Apr 01 21:04:20 CEST 2003 by weigel at tikehau

DBS %> help
help
USAGE:
  java caa.db.DBInterpreter
        interpreter in interactive mode
  java caa.db.DBInterpreter <inputFileName> [<outputFileName>]
        interpreter in batch mode

INTERPRETER COMMANDS:
  help
        prints this help message
  quit
        stops the interpreter and exits (current connection is closed)
  processBatch <inputFileName> <outputFileName>
        processes a batch file, writing the output to the specified file
  connect <dbType> <dbHost> <dbPort> <dbName> <dbLogin> <dbPassword>
        opens a connection to a database system
  disconnect
        closes the current connection to the database system
  createDB <dbName> [<encodingType>]
        creates a new database (current database connection is kept)
```

```
  removeDB <dbName>
         removes a database (and disconnects, if the current database is removed)
  infoDB
         prints out the parameter setting of the current database
  fillDB (@Batch <batchFileName> [<inputFileNames>]|<inputFileNames>])
         reads data into a database, identifying nodes with the specified ID type
  clearDB
         removes all data and indices from a database
  initDB <idScheme> <virtArity> <virtAdjust>
         creates node IDs in a database (removing existing node IDs and indices)
  setResultOutput <resultStyle>
         specifies when to print query results
  queryDB <integration> <iteration> <query>
         [<indexType> <textContain> <descDB> <descIndex> <navIndex>]
         executes a query the specified number of times using the specified index
         (multiple-token queries in 'single quotes')
  infoIndex <infoStyle> <indexType> <textContain> <descDB> <descIndex> <navIndex>
         prints out a serialization of the index with the specified parameters, if there is one
  createIndex <indexType> <textContain> <descDB> <descIndex> <navIndex>
         adds a new index with the specified parameters to a database
  removeIndex <indexType> <textContain> <descDB> <descIndex> <navIndex>
         removes the specified index from a database
  loadIndex <indexType> <textContain> <descDB> <descIndex> <navIndex>
         loads the specified index into main memory
  releaseIndex <indexType> <textContain> <descDB> <descIndex> <navIndex>
         releases references to the specified index in main memory
  readPerformance <indexType> <textContain> <descDB> <descIndex> <navIndex> <maintenance>
         (@Batch <batchFileName> [<inputFileNames>]|<inputFileNames>])
         extracts performance data from query protocol files
         (both absolute and relative to a specified reference index configuration),
         adding/substituting it to previously extracted data
  analyzePerformance <analysis>
         analyzes performance information extracted from query protocol files
  formatPerformance <outputFormat> <outputFileNamePrefix> <formatting>
         writes performance analyses in a specified output format to one or more files
         (a running output file number, beginning with 0,
         and an appropriate file name extension is appended to the given file name automatically)
  analyzeKeywords <numKeywords> <pathSelect> <pathUnselect> <nodeSelect> <nodeUnselect>
         <outputFileName> <indexType> <textContain> <descDB> <descIndex> <navIndex>
         writes keyword path and node selectivity information collected from an index to a file
         (for more information, see the documentation of
         caa.index.navigational.IndexInterface.getXMLKeywordAnalysis())
  generateQueries <integration> <iteration> <classSize> <querySize> <queryDepth> <queryID>
         <dbType> <dbHost> <dbPort> <dbName> <dbLogin> <dbPassword> <resultStyle> <garbageStyle>
         <minValidTrials> <maxValidTrials> <maxTrials> <evalTimeout>
         <dtdFileName> <keywordFileName> <queryInputFileNamePrefix> <queryOutputFileNamePrefix>
         <generatorClass> [<indexType> <textContain> <descDB> <descIndex> <navIndex>]
         generates batch query files for testing all query classes with all index configurations
         (if an index is specified, the generated queries are evaluated on the currently connected
         database to identify mismatches)
         (see the documentation for caa.datastructure.query.QueryGeneratorInterface)
  collectGarbage
         cleans up the memory
where
  <dbType>          = @Postgres
  <encodingType>    = (@ASCII|@Latin1)
  <idScheme>        = (@DepthFirst|@VirtNodesDepth|@BreadthFirst|@VirtNodesBreadth)
  <virtArity>       = (<any positive integer>|-1)
```

```
<virtAdjust>       = (@Tree|@Level|@Path|@None)
<indexType>        = (
                      @DataGuide|
                      @IDCompSingleCADG|
                      @1BLookUpSigSingleCADG|
                      @2BLookUpSigSingleCADG|
                      @4BLookUpSigSingleCADG|
                      @8BLookUpSigSingleCADG|
                      @1BGenerateSigSingleCADG|
                      @2BGenerateSigSingleCADG|
                      @4BGenerateSigSingleCADG|
                      @8BGenerateSigSingleCADG
                     )
<textContain>      = (@Direct|@Indirect)
<descDB>           = (@Interval|@Navigation|@ChildVerification|@None)
<descIndex>        = (@Interval|@None)
<navIndex>         = (@Label|@None)
<integration>      = (@Loose|@Tight)
<infoStyle>        = (@NavStructure|@All)
<resultStyle>      = (@Always|@First|@Never)
<garbageStyle>     = (@Query|@Class|@IndexConfiguration|@Never)
<numKeywords>      = <any positive integer>
<iteration>        = <any positive integer greater than zero>
<classSize>        = <any positive integer greater than zero>
<querySize>        = <any positive integer greater than zero>
<queryDepth>       = <any positive integer greater than zero>
<minValidTrials>   = <any positive integer greater than zero>
<maxValidTrials>   = <any positive integer greater than zero, and not smaller than <minValidTrials>>
<maxTrials>        = <any positive integer greater than zero, and not smaller than <maxValidTrials>>
<evalTimeout>      = <any positive integer greater than zero>
<queryID>          = <any integer>
<pathSelect>       = <any real in the range [0..1]>
<pathUnselect>     = <any real in the range [0..1]>
<nodeSelect>       = <any real in the range [0..1]>
<nodeUnselect>     = <any real in the range [0..1]>
<maintenance>      = (@Add|@Replace)
<outputFormat>     = (
                      @LaTeX (.tex)
                      @GNUPlot (.plt)
                     )
<analysis>  =        (
                      @QueryClass2IndexConf <maxResults>
                      @IndexConf2QueryClass <maxResults>
                      @QueryParam2IndexConf <maxResults>
                      @IndexConf2QueryParam <maxResults>
                      @QueryClass2IndexParam <maxResults>
                      @IndexParam2QueryClass <maxResults>
                      @QueryParam2IndexParam <maxResults>
                      @IndexParam2QueryParam <maxResults>
                     )
<formatting>  =      (
                      LaTeX output:
                      (@StandAlone|@Fragment) @AbsRaw
                      (@StandAlone|@Fragment) @AbsMinClass
                      (@StandAlone|@Fragment) @AbsMinIteration
                      (@StandAlone|@Fragment) @AbsMaxClass
                      (@StandAlone|@Fragment) @AbsMaxIteration
                      (@StandAlone|@Fragment) @AbsAvgClass
```

```
                    (@StandAlone|@Fragment) @AbsAvgIteration
                    (@StandAlone|@Fragment) @AbsDevClass
                    (@StandAlone|@Fragment) @AbsDevIteration
                    (@StandAlone|@Fragment) @AbsVarClass
                    (@StandAlone|@Fragment) @AbsVarIteration
                    (@StandAlone|@Fragment) @RelRaw
                    (@StandAlone|@Fragment) @RelMinClass
                    (@StandAlone|@Fragment) @RelMinIteration
                    (@StandAlone|@Fragment) @RelMaxClass
                    (@StandAlone|@Fragment) @RelMaxIteration
                    (@StandAlone|@Fragment) @RelAvgClass
                    (@StandAlone|@Fragment) @RelAvgIteration
                    (@StandAlone|@Fragment) @RelDevClass
                    (@StandAlone|@Fragment) @RelDevIteration
                    (@StandAlone|@Fragment) @RelVarClass
                    (@StandAlone|@Fragment) @RelVarIteration
                    (@StandAlone|@Fragment) @QueriesInfo

                    GNUPlot output:
                    @Min [<queryClasses>]
                    @Max [<queryClasses>]
                    @Avg [<queryClasses>]
                    @Dev [<queryClasses>]
                    @Var [<queryClasses>]
                   )

For details on database and index parameters, please consult the documentation
of the class caa.db.DBInterface.


DBS %> quit
quit

=======================================
```

Bibliography

[Abi97] Serge Abiteboul. Querying Semi-Structured Data. In *Proceedings of the 6th International Conference on Database Theory (ICDT)*, pages 1–18, 1997.
WWW: http://citeseer.nj.nec.com/abiteboul97querying.html
Local: http://www.pms.informatik.uni-muenchen.de/publikationen/projektarbeiten/Felix.Weigel/xmlindex/material/abiteboul97querying.pdf

[BPSMM00] Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, and Eve Maler. Extensible Markup Language (XML) 1.0 (Second Edition). W3C Recommendation, October 2000.
WWW: http://www.w3.org/TR/REC-xml

[Bun97] Peter Buneman. Semistructured Data. In *Proceedings of the 16th ACM Symposium on Principles of Database Systems*, pages 117–121, 1997.
WWW: http://citeseer.nj.nec.com/buneman97semistructured.html
Local: http://www.pms.informatik.uni-muenchen.de/publikationen/projektarbeiten/Felix.Weigel/xmlindex/material/buneman97ssd.pdf

[CA98] Yangjun Chen and Karl Aberer. Layered Index Structures in Document Database Systems. In *Proceedings of the Seventh International ACM Conference on Information and Knowledge Management (CIKM)*, pages 406–413, November 1998.
WWW: http://xml.darmstadt.gmd.de/cc-publications.html#4
Local: http://www.pms.informatik.uni-muenchen.de/publikationen/projektarbeiten/Felix.Weigel/xmlindex/material/chen98layer.pdf

[CA99] Yangjun Chen and Karl Aberer. Combining Pat-Trees and Signature Files for Query Evaluation in Document Databases. In *Database and Expert Systems Applications*, pages 473–484, 1999.
WWW: http://citeseer.nj.nec.com/390292.html
Local: http://www.pms.informatik.uni-muenchen.de/publikationen/projektarbeiten/Felix.Weigel/xmlindex/material/chen99combining.pdf

[CD99] James Clark and Steve DeRose. XML Path Language (XPath), version 1.0. W3C Recommendation, November 1999.
WWW: http://www.w3.org/TR/xpath

[Coo98] James W. Cooper. *The Design Patterns Java Companion*. Design Patterns Series. Addison-Wesley, 1998.

[CS01] Brian Cooper and Moshe Shadmon. The Index Fabric: Technical Overview. Technical report, RightOrder Inc., 2001.
WWW: http://www.rightorder.com/Technology%5FWhitePaper.html
Local:

[CSF+01] Brian Cooper, Neal Sample, Michael J. Franklin, Gísli R. Hjaltason, and Moshe Shadmon. A Fast Index for Semistructured Data. In *Proceedings of 27th International Conference on Very Large Data Bases (VLDB)*, pages 341–350, 2001.
WWW: http://citeseer.nj.nec.com/cooper01fast.html
Local: http://www.pms.informatik.uni-muenchen.de/publikationen/projektarbeiten/Felix.Weigel/xmlindex/material/cooper01fast.pdf

[CT01] John Cowan and Richard Tobin. XML Information Set. W3C Recommendation, October 2001.
WWW: http://www.w3.org/TR/xml-infoset

[DML98] D. Dervos, Y. Manolopoulos, and P. Linardis. Comparison of Signature File Models with Superimposed Coding. *Information Processing Letters*, 65(2):101–106, 1998.

[Fal85] C. Faloutsos. Signature files: Design and performance comparison of some signature extraction methods. In *Proceedings of the 1985 SIGMOD Conference*, pages 63–82, May 1985.

[FK99] Daniela Florescu and Donald Kossmann. A Performance Evaluation of Alternative Mapping Schemes for Storing XML Data in a Relational Database. Technical Report 3680, INRIA, May 1999.
WWW: http://citeseer.nj.nec.com/florescu99performance.html
Local: http://www.pms.informatik.uni-muenchen.de/publikationen/projektarbeiten/Felix.Weigel/xmlindex/material/florescu99performance.pdf

[Fre60] E. Fredkin. Trie Memory. *Communications of the ACM*, 3(9):490–499, September 1960.

[GMW99] R. Goldman, J. McHugh, and J. Widom. From Semistructured Data to XML: Migrating the Lore Data Model and Query Language. In *Proceedings of the 2nd International Workshop on the Web and Databases (WebDB)*, pages 25–30, 1999.
WWW: http://citeseer.nj.nec.com/goldman99from.html
Local: http://www.pms.informatik.uni-muenchen.de/publikationen/projektarbeiten/Felix.Weigel/xmlindex/material/goldman99mig.pdf

[GW97a] R. Goldman and J. Widom. DataGuides: Enabling Query Formulation and Optimization in Semistructured Databases. In *Proceedings of the Twenty-Third International Conference on Very Large Data Bases (VLDB)*, pages 436–445, 1997.
WWW: http://citeseer.nj.nec.com/126680.html
Local: http://www.pms.informatik.uni-muenchen.de/publikationen/projektarbeiten/Felix.Weigel/xmlindex/material/goldman97dataguides.pdf

[GW97b] R. Goldman and J. Widom. DataGuides: Enabling Query Formulation and Optimization in Semistructured Databases. Technical report, Stanford University, Computer Science Department, Database Group, 1997.
WWW: http://dbpubs.stanford.edu:8090/pub/1997-50
Local:

[JDB] Java Database Connectivity (JDBC). On-line resource.
WWW: http://java.sun.com/products/jdbc

[Kil92] P. Kilpeläinen. *Tree Matching Problems with Applications to Structured Text Databases.* PhD thesis, Dept. of Computer Science, University of Helsinki, 1992.

[LM01] Quanzhong Li and Bongki Moon. Indexing and Querying XML Data for Regular Path Expressions. In *Proceedings of the 27th VLDB Conference*, pages 361–370, 2001.
WWW: http://citeseer.nj.nec.com/454193.html
Local: http://www.pms.informatik.uni-muenchen.de/publikationen/projektarbeiten/Felix.Weigel/xmlindex/material/li01indexing.pdf

[LYYB96] Yong Kyu Lee, Seong-Joon Yoo, Kyoungro Yoon, and P. Bruce Berra. Index structures for structured documents. *Digital Libraries*, pages 91–99, 1996.
WWW: http://citeseer.nj.nec.com/lee96index.html
Local: http://www.pms.informatik.uni-muenchen.de/publikationen/projektarbeiten/Felix.Weigel/xmlindex/material/lee96index.pdf

[MAG+97] Jason McHugh, Serge Abiteboul, Roy Goldman, Dallan Quass, and Jennifer Widom. Lore: A Database Management System for Semistructured Data. *SIGMOD Record*, 26(3):54–66, 1997.
WWW: http://citeseer.nj.nec.com/mchugh97lore.html
Local: http://www.pms.informatik.uni-muenchen.de/publikationen/projektarbeiten/Felix.Weigel/xmlindex/material/mchugh97lore.pdf

[Meu98] Holger Meuss. Indexed tree matching with complete answer representations. In *Workshop on Principles of Digital Document Processing (PODDP)*, page 104ff., 1998.
WWW: http://citeseer.nj.nec.com/meuss98indexed.html
Local: http://www.pms.informatik.uni-muenchen.de/publikationen/projektarbeiten/Felix.Weigel/xmlindex/material/meuss98indexed.pdf

[Meu00] H. Meuss. *Logical Tree Matching with Complete Answer Aggregates for Retrieving Structured Documents.* PhD thesis, Dept. of Computer Science, University of Munich, 2000.

WWW: http://citeseer.nj.nec.com/meuss00logical.html
Local: http://www.pms.informatik.uni-muenchen.de/publikationen/projektarbeiten/Felix.Weigel/xmlindex/material/meuss00logical.pdf

[MFK+00] Ioana Manolescu, Daniela Florescu, Donald Kossmann, Dan Olteanu, and Florian Xhumari. Agora: Living with XML and Relational. In *The VLDB Journal*, pages 623–626, 2000.
WWW: http://citeseer.nj.nec.com/manolescu00agora.html
Local: http://www.pms.informatik.uni-muenchen.de/publikationen/projektarbeiten/Felix.Weigel/xmlindex/material/manolescu00agora.pdf

[Mor68] D. R. Morrison. PATRICIA: Practical Algorithm To Retrieve Information Coded In Alphanumeric. *Journal of the ACM*, 15(4):514–534, 1968.

[MS99a] H. Meuss and C. Strohmaier. Improving index structures for structured document retrieval. In *Proceedings of the 21st Annual Colloquium on IR Research (IRSG)*, 1999.
WWW: http://citeseer.nj.nec.com/meuss99improving.html
Local: http://www.pms.informatik.uni-muenchen.de/publikationen/projektarbeiten/Felix.Weigel/xmlindex/material/meuss99impr.pdf

[MS99b] Tova Milo and Dan Suciu. Index Structures for Path Expressions. In *Proceedings of the 7th International Conference on Database Theory (ICDT)*, pages 277–295, 1999.
WWW: http://citeseer.nj.nec.com/milo97index.html
Local: http://www.pms.informatik.uni-muenchen.de/publikationen/projektarbeiten/Felix.Weigel/xmlindex/material/milo97index.pdf

[MS01] Holger Meuss and Klaus U. Schulz. Complete Answer Aggregates for Tree-like Databases: A Novel Approach to Combine Querying and Navigation. *ACM Transactions on Information Systems (TOIS)*, 19(2):161–215, April 2001.
WWW: http://citeseer.nj.nec.com/meuss01complete.html
Local: http://www.pms.informatik.uni-muenchen.de/publikationen/projektarbeiten/Felix.Weigel/xmlindex/material/meuss01complete.pdf

[NBY97] Gonzalo Navarro and Ricardo A. Baeza-Yates. Proximal Nodes: A Model to Query Document Databases by Content and Structure. *Information Systems*, 15(4):400–435, 1997.
WWW: http://citeseer.nj.nec.com/navarro97proximal.html
Local: http://www.pms.informatik.uni-muenchen.de/publikationen/projektarbeiten/Felix.Weigel/xmlindex/material/navarro97prox.pdf

[OMM98] Jürgen Oesterle and Petra Maier-Meyer. The GNoP (German Noun Phrase) Treebank. In *First International Conference on Language Resources and Evaluation*, pages 699–703, 1998.

[PDP02] The PostgreSQL Global Development Group. *PostgreSQL 7.4devel Programmer's Guide*, 1996-2002.
WWW: http://developer.postgresql.org/docs/postgres/programmer.html

[PGMW95] Y. Papakonstantinou, H. Garcia-Molina, and J. Widom. Object exchange across heterogenous information sources. In *Proceedings of the IEEE International Conference on Data Engineering (ICDE)*, volume 11, pages 251–260, 1995.

[PSQ] The PostgreSQL relational database system. On-line resource.
WWW: http://www.postgresql.org

[SCF+02] Neal Sample, Brian Cooper, Michael J. Franklin, Gísli R. Hjaltason, and Levy Cohen. Managing Complex and Varied Data with the IndexFabric, 2002.
WWW: http://citeseer.nj.nec.com/497317.html
Local: http://www.pms.informatik.uni-muenchen.de/publikationen/projektarbeiten/Felix.Weigel/xmlindex/material/sample02managing.pdf

[Sch02] Marcus Schilling. A Graphical Query Language for XML based on Complete Answer Aggregates. Diploma Thesis, University of Munich, Computer Science Institute, Teaching and Research Unit "Programming and Modelling Languages", 2002.
WWW: http://www.pms.informatik.uni-muenchen.de/publikationen/#DA:Marcus.Schilling
Local: http://www.pms.informatik.uni-muenchen.de/publikationen/projektarbeiten/Felix.Weigel/xmlindex/material/schilling02graphical.ps.gz

[SGM86] Standard Generalized Markup Language (SGML). ISO/IEC 8879:1986, 1986.
WWW: http://www.iso.org

[Shi01] Dongwook Shin. XML Indexing and Retrieval with a Hybrid Storage Model. *Knowledge and Information Systems*, 3(2):252–261, 2001.
WWW: http://win-www.uia.ac.be/u/s975420/local/indexhybrid.pdf
Local: http://www.pms.informatik.uni-muenchen.de/publikationen/projektarbeiten/Felix.Weigel/xmlindex/material/shin01hybrid.pdf

[SJJ98] Dongwook Shin, Hyuncheol Jang, and Honglan Jin. BUS: An Effective Indexing and Retrieval Scheme in Structured Documents. In *Proceedings of the 3rd ACM International Conference on Digital Libraries*, pages 235–243, 1998.
WWW: http://win-www.uia.ac.be/u/s975420/local/bus.pdf
Local: http://www.pms.informatik.uni-muenchen.de/publikationen/projektarbeiten/Felix.Weigel/xmlindex/material/shin98bus.pdf

[SKWW01] Albrecht Schmidt, Martin Kersten, Menzo Windhouwer, and Florian Waas. Efficient Relational Storage and Retrieval of XML Documents. *Lecture Notes in Computer Science*, 1997:137ff., 2001.
WWW: http://citeseer.nj.nec.com/schmidt00efficient.html
Local: http://www.pms.informatik.uni-muenchen.de/publikationen/projektarbeiten/Felix.Weigel/xmlindex/material/schmidt00efficient.pdf

[SM83] Gerard Salton and Michael J. McGill. *Introduction to Modern Information Retrieval*. McGraw-Hill, 1983.

[SM99] Christian Strohmaier and Holger Meuss. A Filter for Structured Document Retrieval. Technical Report 99-123, Centrum für Informations- und Sprachverarbeitung (CIS), University of Munich, Germany, 1999.
WWW: http://citeseer.nj.nec.com/220821.html
Local: http://www.pms.informatik.uni-muenchen.de/publikationen/projektarbeiten/Felix.Weigel/xmlindex/material/strohm99filter.pdf

[STZ+99] Jayavel Shanmugasundaram, Kristin Tufte, Chun Zhang, Gang He, David J. DeWitt, and Jeffrey F. Naughton. Relational Databases for Querying XML Documents: Limitations and Opportunities. In Malcolm P. Atkinson, Maria E. Orlowska, Patrick Valduriez, Stanley B. Zdonik, and Michael L. Brodie, editors, *Proceedings of 25th International Conference on Very Large Data Bases (VLDB)*, pages 302–314, 1999.
WWW: http://www.cs.cornell.edu/people/jai/papers/RdbmsForXML.pdf
Local: http://www.pms.informatik.uni-muenchen.de/publikationen/projektarbeiten/Felix.Weigel/xmlindex/material/shanmu99rdbs.pdf

[Suc98] Dan Suciu. An Overview of Semistructured Data. *SIGACT News (ACM Special Interest Group on Automata and Computability Theory)*, 29(4):28–38, 1998.
WWW: http://citeseer.nj.nec.com/160105.html
Local: http://www.pms.informatik.uni-muenchen.de/publikationen/projektarbeiten/Felix.Weigel/xmlindex/material/suciu98ssd.pdf

[SYU99] Takeyuki Shimura, Masatoshi Yoshikawa, and Shunsuke Uemura. Storage and Retrieval of XML Documents Using Object-Relational Databases. In *Database and Expert Systems Applications*, pages 206–217, 1999.
WWW: http://citeseer.nj.nec.com/shimura99storage.html
Local: http://www.pms.informatik.uni-muenchen.de/publikationen/projektarbeiten/Felix.Weigel/xmlindex/material/shimura99storage.pdf

[Vir] VirtuGrade Course in Information Retrieval. On-line tutorial. University of Tübingen (Germany), in cooperation with the Katholieke Universiteit Brabant in Tilburg (Netherlands) and the University of Munich (Germany).
WWW: http://pi0959.kub.nl/Paai/Onderw/Vir/index.html

[vR79] C. J. van Rijsbergen. *Information Retrieval*. Butterworths, 1979.
WWW: http://www.dcs.gla.ac.uk/Keith/Preface.html

[Wei02] Felix Weigel. A Survey of Indexing Techniques for Semistructured Documents. Project Thesis, University of Munich, Computer Science Institute, Teaching and Research Unit "Programming and Modelling Languages", August 2002.
WWW: http://www.pms.informatik.uni-muenchen.de/publikationen/#PA:Felix.Weigel
Local: http://www.pms.informatik.uni-muenchen.de/publikationen/projektarbeiten/Felix.Weigel/xmlindex/material/weigel02survey.pdf

[WLOT01] Wen Qiang Wang, Mong-Li Lee, Beng Chin Ooi, and Kian-Lee Tan. XStorM: A Scalable Storage Mapping Scheme for XML Data. *World Wide Web*, 4(1-2):101–119, 2001.
WWW: http://citeseer.nj.nec.com/488036.html
Local: http://www.pms.informatik.uni-muenchen.de/publikationen/projektarbeiten/Felix.Weigel/xmlindex/material/wang01xstorm.pdf

[XML] XML Benchmark Project. On-line resource. Benchmark suite for XML repositories, provided by CWI Amsterdam, INRIA, Microsoft Inc., and BEA Systems.
WWW: http://monetdb.cwi.nl/xml