A probabilistic XML database on top of MayBMS

Paul Stapersma
University of Twente
P.O. Box 217, 7500AE Enschede
The Netherlands
p.stapersma-1@student.utwente.nl

ABSTRACT

Stored data is assumed to be correct, but in some contexts, like data integration, data cleaning or handling sensor data, this is not always the case. We have to cope with uncertainty in the data. Uncertain table structured data can be stored in uncertain relational databases (URDBMS). Unfortunately, in many cases information is represented in a semi structured way. As a result, various probabilistic XML models have been proposed to store uncertain data.

One disadvantage of the XML data model is that queries cannot be executed as efficiently as in the relational database model. Many non-probabilistic mapping techniques have been developed to map semi structured data into relational databases to overcome this disadvantage.

In this research, we use the probabilistic XML model proposed in Van Keulen en De Keijzer [22] in combination with the schema-less mapping technique ‘XPath Accelerator’ to build a probabilistic XML database (PXML-DBMS) based on an URDBMS. A working prototype can be found at http://code.google.com/p/pxmlconverter/.

Keywords

Probabilistic XML, Uncertain Relational Database (URDBMS), XPath, Uncertain semi structured Data, MayBMS

1. INTRODUCTION

Correctness of stored data cannot always be guaranteed. For example, data can be subjected to measurement inaccuracy, ageing, wrong user input, etc. When we store uncertain data, it is desirable to label the data as ‘uncertain’ together with information about the uncertainty in terms of probability. This is called probabilistic data.

Probabilistic data makes it possible to store probabilistic, uncertain, incomplete, fuzzy and imprecise information. Probabilistic data can be stored in relational databases, called uncertain relational databases or in a probabilistic XML database.

Our goal is to build a prototype probabilistic XML database that can store probabilistic XML documents and evaluate XPath queries on probabilistic XML documents both in a scalable manner. Van Keulen en De Keijzer propose the following approaches [22]:

1. The first approach is to extend an XML-DBMS with support for uncertainty.

2. The second approach is to extend an URDBMS with support for XML. There are roughly two important approaches known to accomplish this:

2a. A schema-less approach: probabilistic XML is mapped into an URDBMS without a schema that describes the probabilistic XML content.

2b. A schema-based approach: probabilistic XML is mapped into an URDBMS, based on a schema that describes the probabilistic XML content.

Also the choice of the URDBMS plays an important role. Several URDBMSs have been developed, which take different approaches to store uncertain relational data. Examples are Trio[19, 5, 3, 23], MayBMS[18, 4], Mystiq[6] and Orion[8].

This research is part of a larger research effort to investigate which of the above approaches provide the most scalability.

In this paper, we focus on approach 2a to build our prototype in combination with MayBMS, which we choose as a representative URDBMS. To teach MayBMS how to handle probabilistic XML data, we use ‘XPath Accelerator’ as representative schema-less mapping technique, which we extend for mapping probabilistic XML to MayBMS. We investigate the properties and behaviour of the prototype by experimenting with different kinds of queries on varying amounts of data containing varying amounts of uncertainty.

2. RELATED WORK

Hollander and Van Keulen [13] investigated the behaviour and properties of approach 2a with ‘XPath Accelerator’ and approach 2b with ‘Shared Inlining’, both in combination with ‘Trio’. Their results show that the ‘XPath Accelerator’-based approach was much less efficient than their ‘Inlining’-based approach. They observed that Trio’s confidence calculation was relatively inexpensive. A second observation was that queries scaled exponentially when adopting the ‘XPath Accelerator’ technique.

De Keijzer and Van Keulen [22] used approach 1 for their probabilistic data integration research. They fully support an XML DBMS by adding functionality of handling uncertainty. No results on scalability reported.
3. PROBABILISTIC DATABASES

In this section, we discuss the possible world theory that forms the basis for storing uncertain data, followed by the way probabilistic XML and MayBMS store uncertain data and how to query this uncertain data. This is the background for constructing our extended mapping technique to map probabilistic XML to MayBMS.

3.1 Possible world theory

Probabilistic XML databases and uncertain relational databases are based on the possible world theory as illustrated in figure 1. A possible world is a possible representation for a set of entities. Given a set of possible worlds, an entity can have more than one representation. Therefore, a query on a set of possible worlds results in a set of possible answers.

3.2 Probabilistic XML

Various probabilistic XML models have been proposed [22, 16, 2, 15, 9, 17, 20, 21, 1]. We use the probabilistic XML model proposed in [22].

Probabilistic XML is well-formed XML with an extension for representing uncertainty. This extension is based on two new node kinds: probability nodes (\(\forall\)) to represent choice points; different places in the XML-tree where different possibilities can occur and possibility nodes (\(\forall\)) to represent the alternatives with associated probability.

As example, figure 2 shows a probabilistic XML-tree of weather reports. In this case, only the weather reports of London and Enschede are available for today. The node ‘Today’ has two probability node children. Both probability nodes have two possibility nodes, representing two possibilities. In case of the first probability node, we obtain the possibility ‘Enschede’ with content ‘Dry’ and the possibility ‘Enschede’ with content ‘Wet’. Both possibilities have a probability of 0.5 to be correct. The same applies to the other probability node. Two independent probability nodes with both two possibilities represent four possible worlds.

3.3 Query on probabilistic XML

XPath is a query language for selecting nodes from an XML document. Probabilistic XML is well-formed XML; it conforms to the XML syntax. Therefore, we use XPath as query language for probabilistic XML documents.

Probabilistic XML represent a set of possible worlds. Therefore, an XPath query on a probabilistic XML document returns a set of possible answers. This result is returned as probabilistic XML fragments.

3.4 Uncertain RDBMS

Uncertain relational databases also represent a set of possible worlds. In MayBMS, this is done with the use of a finite set of independent random variables with finite domains and associated probability in a so-called world tableau. Random variables with associated value are assigned to entries in normal relational tables, which then become dependent on those variables.

Random variables are created and assigned with the ‘repair key’-statement. For this statement, a ‘key’-column and a ‘weight’-column must be specified. MayBMS tries to create a unique primary key for every entry in the processed table. For every key that violates the uniqueness constraint, MayBMS creates an independent random variable and assigns it with an unique value and corresponding weight per entry to the entries that have the same key. Per random variable, the sum of all weights per possible value is 1. The entries with the same duplicate key are subsequently dependent on the same variable with different values [14].

We illustrate this process with figure 3 and the ‘repair key’-statement “REPAIR KEY ‘Key’ IN R WEIGHT by ‘Weight’”. Table ‘R’ has three duplicate keys in the ‘Key’-column. For every duplicate key, an independent random variable is made with unique value per entry and corresponding weight. For the first three entries, the variable \(a\) is created and assigned with unique values to these entries. The weight of the assignments of variable \(a\) corresponds to the value of the ‘Weight’-column divided by the sum of the weight of all the entries variable \(a\) is assigned to. In this case, the first entry has weight \(2/(2+3+5) = 0.2\), the second entry \(3/(2+3+5) = 0.3\) and so forth. The result of the ‘repair key’-statement is illustrated in figure 4.

3.5 URDBMS query

MayBMS uses SPROUT as query engine which extends the query engine of PostgreSQL with a new physical aggregation operator for confidence computation. SPROUT supports the I-SQL language, which consists of a subset of the SQL language and extra operations. Like probabilistic XML queries, queries on an URDBMS also return a set of possible answers, expressed in uncertain tables [4, 14].
4. PROBLEM STATEMENT

Many applications are developed to store uncertain data in a relational way. Unfortunately, in many cases information is represented in a semi structured way. Various probabilistic XML models have been proposed to store uncertain data in the semi structured data model, but as far as we know, ProTDB [20] is the only database that can store probabilistic data in a probabilistic XML database. The ProTDB database is based on an XML database (approach 1 of section 1).

We build a probabilistic XML database on top of an URDBMS, based on a schema-less mapping technique. This is one approach Van Keulen en De Keijzer [22] describe to store and query probabilistix XML, see section 1. No system exists that uses XML with MayBMS.

The following sections cover questions about storing semi structured data in a relational data model and preserving the same set of possible world in the semi-structured data model as in the relational data model.

4.1 Mapping PXML into URDBMS

Probabilistic XML and URDBMS both store a set of possible worlds, but store them in different ways, as described in section 3. When we store probabilistic XML in an URDBMS, we must guarantee that the set of possible worlds in probabilistic XML is the same as the set of possible worlds in the URDBMS. Therefore, we construct a mapping from probabilistic XML to URDBMS that guarantees this.

Many techniques have been developed for mapping XML to RDBMS-systems and querying XML data using RDBMS-systems. These techniques can roughly be grouped into techniques that require an XML-schema and techniques that do not require an XML-schema. In this research, we focus on a schema-less approach. One useful property of semi structured data, is that data doesn’t have to conform to a schema. With a schema-based approach, we lose this property and are permitted to include a schema for every XML document we wish to store. This was our motivation to choose for a schema-less approach.

4.2 Goal

The goal of this research is to build a probabilistic XML-DBMS that can store probabilistic XML documents using a schema-less mapping technique and evaluate XPath queries on probabilistic XML documents both in a scalable manner. We intend to use this database for large amounts of probabilistic XML data with different kinds of content. Our proposed probabilistic probabilistic XML database must guarantee that the set of possible worlds is preserved in MayBMS.

4.3 Approach

Figure 5 shows the general architecture we use for this research. We use MayBMS as a representative URDBMS and implement the converters in Java.
of possibility nodes. This allows us to associate ‘accel’-entries with a set of
node. The result can be found in figure 12.

Don’t care choice points.
We must take into account the situation in which an XML node
always exists, like node ‘a’ in figure 6. To handle this situation, we put the whole probabilistic XML document inside a possibility node with \( \text{prob} = 1.0 \), so that every entry in table ‘accel’ has a preceding possibility node. The corresponding entry of this new possibility node can also be used for a ‘don’t care’-assignment, because it is a single child.\(^1\)

Phase join.
XML nodes can have multiple ancestor possibility nodes. This requires multiple joins with tables ‘accel’ and ‘ucp’. In the most optimised situation, we need to join as many times as the maximum number of dependencies an XML node in the document has. This corresponds to the maximum number of ancestor possibility nodes the XML node with the most ancestor possibility nodes has.

When two possibility nodes are independent of each other, that is to say, they are not descendants or ancestors of each other, we can join these nodes simultaneously with the table ‘accel’, because the descendant XML nodes of the first possibility node do not depend on the second possibility node. Therefore, we split the possibility nodes into phases. A phase contains all independent possibility nodes that are not in another phase.

The phase column in figure 9 is filled with the number of preceding possibility nodes per possibility node. This ensures that the preceding possibility nodes inside the same phase are independent of each other. This is illustrated in figure 11. Every level of possibility nodes has a sepa-

\(^1\) A single child possibility node is always chosen. We therefore conclude that a join with a single child possibility node results in no dependency for the ‘accel’-entry which is being joined.
Figure 11. The probabilistic XML tree of figure 6 after adding a new root node and defining phases.

Figure 12. The resulting uncertain accel table associated with figure 6

rate phase. In ‘phase = 0’, we join all XML nodes with the root probability node. The possibility nodes with prob = 0.6, prob = 0.4 are the ‘phase = 1’-possibility nodes. In ‘phase = 1’, we join these nodes with their descendants. In ‘phase = 2’, we join the ‘phase = 2’ possibility nodes with their descendants. We omit the ‘phase = 0’-join, because this phase only has one ‘don’t care’-entry in the ‘cp’-table (the root possibility node).

Notice that a phase join doesn’t have to result in a join with all the entries in the ‘accel’-table. The ‘accel’-entries that do not have an ancestor possibility node in the processed phase do not depend on a possibility node in the processed phase. We therefore join these nodes with the ‘don’t care’-entry of the first phase.

For example, if we look at the ‘phase = 2’-join in figure 11, we join node ‘e’ with ‘prob = 0.3’-possibility node, node ‘f’ with ‘prob = 0.7’-possibility node, node ‘g’ with ‘prob = 0.2’-possibility node, node ‘h’ with ‘prob = 0.8’-possibility node and all the other nodes with the ‘prob = 1.0’-possibility node of the first phase.

Result.

After all phase joins are completed, we have obtained table ‘accel’. Figure 12 shows the ‘accel’ table when we use figure 6 as input. Table ‘accel’ stores the same set of possible worlds as the probabilistic XML representation, that is to say, the existence of every entry is dependent on the exact same choices as in the probabilistic XML document.

5.1 Querying probabilistic XML with XPath

For the probabilistic XML database, we use XPath as query-language and MayBMS for storing possible worlds. MayBMS uses I-SQL as query language. We use the standard XPath to SQL conversion for mapping XPath to I-SQL as described in [10]. This approach is valid, because I-SQL supports a subset of SQL. We do not have to modify this query mapping technique, because a normal query in combination with a set of possible worlds results in a set of possible answers according to the possible world theory, see section 3.1. The result of a converted XPath query is an URDBMS-table, which we convert to a probabilistic XML-representation. This probabilistic XML-representation is the result the probabilistic XML database returns for the XPath query. In this research, we omit the conversion of URDBMS-tables to probabilistic XML.

6. EXPERIMENTAL SETUP

6.1 Goal and testset description

In this experiment, we investigate the effects on the execution time of queries when we add probability to data sets and change the amount of XML elements in data sets. We use 7 different XML movie data sets gathered from the Internet Movie Database (http://www.imdb.com). All these data sets conform to the XML structure in figure 15. The data sets have different amounts of XML elements. Smaller data sets are subsets of larger data sets, whereby all movie entries are complete.

6.2 Add uncertainty to data sets

We add several levels of uncertainty to the 7 XML movie data sets by evaluating an operation for every XML element. This operation ‘addProb’ processes all children of the processed node and returns a subset of children based on a given probability-parameter. These children are put in different possible worlds.

An example can be found in figure 13. In this figure, ‘addProb’ returns the children ‘a2’ and ‘a3’ based on a given probability. To add uncertainty, we insert a possibility node as parent for these nodes and insert one probability node as grandparent. Finally, we assign a probability to the possibility nodes, whereby the total probability of 1 is distributed among the inserted possibility nodes. In this case, we assign prob = 0.5 to both possibility nodes.

Actual probabilities have no impact on query execution time in MayBMS, so we are allowed to equally distribute the total probability of 1 among the uncertain children without consequences.

6.3 XPath queries

We use 14 queries as representatives for most XPath categories, which can be found in figure 14.

We describe the XPath queries shortly: Q1 retrieves the year of an existing movie. Q2 tries to do the same for a non-existing movie. Q3 returns all keywords of the movie ‘Arriving Tuesday’. Q4 retrieves all movies where one of its directors is ‘Riddiford, Richard’. Q5 returns the movies that use the location ‘Perth, Western Australia, Australia’. Q6 retrieves all movies where one of its directors is ‘Verhinski, Gore’, but takes a different approach then Q4. Q7 returns the name of the person with the role ‘Opossum Man’. Q8 and Q9 return all the roles associated with ‘Essel, Franz’ or ‘Smyth, Jon’. Q10 returns the following genres of the genre ‘Horror’ for the movie ‘Muertos no hablan, Los’. Q11 retrieves all movies of the same year
as the movies in which ‘Davenport, Jack’ is actor. Q12 returns all movies of the same year as the movies in which ‘Bustillo Oro, Jean’ is director. Q13 returns all movies of the year 2000. Q14 returns all directors.

These queries are translated into I-SQL queries as described in [10]. The query execution time is determined by the time it takes to execute the converted XPath query. The time it takes to convert an XPath query into an I-SQL query is not be taken into account, because we are only interested in the performance of the URDBMS in combination with our mapping technique.

### 6.4 Test platform

All testing was performed on a MacBook Pro with 4 GB 1067 MHz DDR3, 3.06 GHz Intel Core 2 Duo processor, SSD harddrive (APPLE SSD TS256A) and Mac OS X version 10.6.4 as operating system. We use the average execution time of the I-SQL queries over the last 5 runs. We executed 6 runs to ensure the database was hot.

### 7. EXPERIMENTAL RESULTS

Our experiments focus on the examination of key aspects of scalability. We compare the scalability of queries on both certain and uncertain data sets. We also compare the scalability of both methods for increasing data sets.

We use a set of 14 XPath queries, which can be found in figure 14. The test results for queries 3, 9, 10, 13 and 14 can be found in figures 16, 18, 19, 20 and 21.

#### 7.1 Scalability in data volume

We examine the scalability of different XPath queries on increasing data set size.

The test results for queries 7, 8 and 9 show linear time to execute for certain data sets and uncertain data sets, illustrated in Figure 18. All three queries start with a predicate followed by a sibling retrieval. Based on these results, we conclude that the execution time of a sibling retrieval or a predicate retrieval is at most in linear time, disregarding uncertainty.

The test results for queries 1, 3, 4, 5, 6, 13 and 14 show linear time to execute for certain data sets, illustrated in Figures 16, 20 and 21. These queries show odd behaviour when executed on uncertain data sets. Note that for query 5 (and query 1 and 6 as well) the largest data sets show linear time and have in most cases lower execution time then smaller data sets.

This strange behaviour in combination with several outliers in other bar graphs are the result of executing different query execution plans. We matched several odd cases, where we took query execution plans for outliers and compared these plans with query execution plans for same queries on data sets with equal uncertainty and same queries on same size data sets. In all cases, different query execution plans were executed. The same behaviour for certain data sets is observed in [7]. The query optimizer occasionally estimates an incorrect number of rows, which can result in a bad query execution plan and high execution time. This problem lies with PostgreSQL on which MayBMS is build.

When correct query plans would have been created, we presume that queries 1, 3, 4, 5, 6, 13 and 14 on uncertain data sets would also have shown execution in linear time, because certain data sets show this behaviour as well as the larger uncertain data sets.

The test results of query 2 show a short query execution time independent of the data set size when outliers are ignored. Query 2 tries to retrieve the year of a movie that doesn’t exists. These results indicate that determining the existence of an XML element based on its content is very scalable. We have to remark that we use relative small XML data sets.

#### 7.2 Influence of the amount of uncertainty

We examine the scalability of different XPath queries on increasing amount of uncertainty.

Table 2 shows for queries in the same execution time range the average execution time per data set per query\(^2\), outliers included. This figure does not indicate an influence of the query execution time and the amount of uncertainty, excluding queries on certain data sets.

Test results for queries 1, 2, 5-9, 14 show clusters for equal amount of data of equal height which also indicate that there is no relation between the amount of uncertainty and the execution time, also excluding queries on certain data sets.

On average, query execution on certain data sets is 453 times as fast as on uncertain data sets. These numbers indicate a very ineffective approach for query execution on uncertain data sets.

Table 2 shows the average execution time on certain data sets and uncertain data sets. Query execution on certain data sets is in all cases much faster, unregarded the executed query. Most execution times on uncertain data sets are twice as high as execution times on certain data sets. Worst case query execution times are in all cases more than 6 times the average query execution time on uncertain data sets for the same query. In case of query 4 this is even 6055 times the average query execution time.

8. CONCLUSIONS

This work has been primarily motivated by the need for an probabilistic XML database based on an URDBMS. We extended the ‘XPath Accelerator’ mapping approach in order to build a probabilistic XML database based on an URDBMS. Test results for this prototype database indicate high query execution times, caused by (1) inefficient query execution plans, described in section 7.1 and (2) handling uncertainty, as described in section 7.3. Although the test results indicate a non-scalable system, we are convinced that an improved query engine could highly contribute to a more efficient system for the use of XML with MayBMS.

### 9. FUTURE WORK

\(^2\)The average execution time is calculated by the sum of all execution times of the same query executed on data sets with equal uncertainty.
This work covers approach 2a, described in section 1. Further investigation of approaches 1 and 2b on top of MayBMS, as well as a final comparison of the three approaches is needed.

The influence of the amount of phases is not investigated. One extra phase results in three extra columns in MayBMS. The used testsets do not have deep tree structures, which result in a low variety of phases. Therefore, the influence of phases is not covered in this research.

10. ACKNOWLEDGMENTS

I’d like to acknowledge Maurice van Keulen, my supervisor of this Bachelor Referaat project.

11. REFERENCES


[Exact Match]
( Query 1 ) /movie[title='Prisoner of Zenda']/year/text()
( Query 2 ) /movie[title='ThisMovieDoesn'tExist']/year/text()

[Tree Navigation : covered axis: descendant, ancestor, preceding-sibling, following-sibling]
( Query 3 ) /movie[title/text() = 'Arriving Tuesday']/descendant::keyword/text()
( Query 4 ) /movie[directors/director/text() = 'Riddford, Richard']/descendant::title/text()
( Query 5 ) /location/text() = 'Perth, Western Australia, Australia'/ancestor::movie/title/text()
( Query 6 ) /director/text() = 'Verbinski, Gore'/ancestor::movie/title/text()
( Query 7 ) /role/text() = 'Opossum Man'/preceding-sibling::name/text()
( Query 8 ) /name/text() = 'Essel, Franz'/following-sibling::role/text()
( Query 9 ) /name/text() = 'Smyth, Jon'/following-sibling::role/text()
( Query 10 ) /movie[title/text() = 'Muertos no hablan, Los']/genres/genre/text() = 'Horror'/following-sibling::genre/text()

[Join]
( Query 11 ) /movie/year/movie[actors/actor/name='Davenport, Jack']/year/title/text()
( Query 12 ) /movie/year/movie[directors/director/text() = 'Bustillo Oro, Juan']/year/title/text()
( Query 13 ) /movie/year/text() = '2000'/title/text()
( Query 14 ) /director/text()

Figure 14. Test queries

Figure 15. DTD-graph of the XML document structure

Figure 17. Test results for query 5
Figure 18. Test results for query 9

Figure 19. Test results for query 10

Figure 20. Test results for query 13

Figure 21. Test results for query 14

Figure 22. Test results for query 5 for increasing path length
Table 1. AVG query execution time per data set

<table>
<thead>
<tr>
<th>Uncertainty</th>
<th>0%</th>
<th>10%</th>
<th>20%</th>
<th>30%</th>
<th>40%</th>
<th>50%</th>
<th>60%</th>
<th>70%</th>
<th>80%</th>
<th>90%</th>
<th>100%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Query1</td>
<td>16</td>
<td>60</td>
<td>54</td>
<td>246</td>
<td>338</td>
<td>799</td>
<td>54</td>
<td>262</td>
<td>400</td>
<td>264</td>
<td>259</td>
</tr>
<tr>
<td>Query2</td>
<td>3</td>
<td>27</td>
<td>15</td>
<td>270</td>
<td>319</td>
<td>412</td>
<td>15</td>
<td>39</td>
<td>385</td>
<td>265</td>
<td>266</td>
</tr>
<tr>
<td>Query3</td>
<td>18</td>
<td>57</td>
<td>19307</td>
<td>16677</td>
<td>318535</td>
<td>15927</td>
<td>19307</td>
<td>13210</td>
<td>12756</td>
<td>12344</td>
<td>5873</td>
</tr>
<tr>
<td>Query4</td>
<td>50</td>
<td>8857</td>
<td>11038</td>
<td>20073</td>
<td>17791</td>
<td>250373</td>
<td>11038</td>
<td>29512</td>
<td>238158</td>
<td>16335</td>
<td>5194</td>
</tr>
<tr>
<td>Query5</td>
<td>19</td>
<td>57</td>
<td>302</td>
<td>300</td>
<td>228</td>
<td>226</td>
<td>302</td>
<td>147</td>
<td>288</td>
<td>72</td>
<td>61</td>
</tr>
<tr>
<td>Query6</td>
<td>17</td>
<td>50</td>
<td>256</td>
<td>316</td>
<td>230</td>
<td>326</td>
<td>256</td>
<td>144</td>
<td>126</td>
<td>62</td>
<td>50</td>
</tr>
<tr>
<td>Query7</td>
<td>10</td>
<td>29</td>
<td>29</td>
<td>30</td>
<td>30</td>
<td>29</td>
<td>29</td>
<td>27</td>
<td>24</td>
<td>22</td>
<td></td>
</tr>
<tr>
<td>Query8</td>
<td>17</td>
<td>30</td>
<td>30</td>
<td>29</td>
<td>30</td>
<td>26</td>
<td>30</td>
<td>28</td>
<td>23</td>
<td>23</td>
<td></td>
</tr>
<tr>
<td>Query9</td>
<td>17</td>
<td>29</td>
<td>29</td>
<td>30</td>
<td>29</td>
<td>29</td>
<td>29</td>
<td>25</td>
<td>27</td>
<td>24</td>
<td></td>
</tr>
<tr>
<td>Query13</td>
<td>84</td>
<td>408</td>
<td>498</td>
<td>861</td>
<td>1001</td>
<td>4058</td>
<td>498</td>
<td>2636</td>
<td>2418</td>
<td>893</td>
<td>891</td>
</tr>
<tr>
<td>Query14</td>
<td>14</td>
<td>24</td>
<td>22</td>
<td>22</td>
<td>24</td>
<td>22</td>
<td>22</td>
<td>22</td>
<td>22</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 2. AVG execution times per category

<table>
<thead>
<tr>
<th>Category</th>
<th>Query</th>
<th>Avg Certain</th>
<th>Avg Uncertain</th>
<th>Worst Case</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exact Match</td>
<td>1</td>
<td>16</td>
<td>283</td>
<td>3205</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>3</td>
<td>221</td>
<td>2205</td>
</tr>
<tr>
<td>Tree Navigation</td>
<td>3</td>
<td>16</td>
<td>43489</td>
<td>108991</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>50</td>
<td>60837</td>
<td>127501</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>19</td>
<td>198</td>
<td>1942</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>17</td>
<td>182</td>
<td>1833</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>10</td>
<td>27</td>
<td>112</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>17</td>
<td>27</td>
<td>110</td>
</tr>
<tr>
<td></td>
<td>9</td>
<td>17</td>
<td>27</td>
<td>106</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>-</td>
<td>-</td>
<td>&gt;24h</td>
</tr>
<tr>
<td>Join</td>
<td>11</td>
<td>-</td>
<td>-</td>
<td>&gt;24h</td>
</tr>
<tr>
<td></td>
<td>12</td>
<td>-</td>
<td>-</td>
<td>&gt;24h</td>
</tr>
<tr>
<td>Multiple Entries</td>
<td>13</td>
<td>84</td>
<td>14762</td>
<td>3022</td>
</tr>
<tr>
<td></td>
<td>14</td>
<td>14</td>
<td>23</td>
<td>130</td>
</tr>
</tbody>
</table>

Figure 23. AVG execution times per uncertain data set