Computational complexity of schema matching approaches

PETER MARTINEK, BELA SZIKORA
Department of Electronics Technology
Budapest University of Technology and Economics
Goldman Gy. tér 3., 1111 Budapest
HUNGARY
martinek@ett.bme.hu

Abstract: - Comparing and integrating of different data structures e.g. relational databases of information systems is a current problem in information sciences. Various solutions have appeared in the last 10 years aimed to achieve a high accuracy level in schema integration and similarity measurement of entities originating from different schemas. The capabilities of approaches are usually properly evaluated from the point of view of accuracy. However the computational complexity of the proposed algorithms is hardly ever examined in these works. We claim that efficiency of a proposal can only be measured by taking into account both the accuracy and the computational requirements of participating algorithms. Since there are many known measurement methods and metrics for the evaluation of accuracy, the focus is set for the analysis of their computational complexity in this paper. After the problem formulation the main ideas behind our method are presented briefly. Many kinds of approximation techniques and applied algorithm theory are used to evaluate different approaches. Three specific approaches were also selected to present the work of our method in details on them. Experiments run on various test inputs are also included.

Key-Words: - Computational complexity, Schema matching, Approximation techniques in computational requirement estimation

1 Introduction
Integration of different software systems and corresponding data structures is a certain need in information technology. The field of its applications is various starting from medical systems to enterprise application integration through E-commerce and E-government. Because system integration is hardly separable from the mapping of different data structures, the schema matching problem i.e. comparing and aligning of entities of schemas to each other is a current issue as well.

The aim of schema matching algorithms is to identify the semantic relevance between the entities of different data representation structures coming from different systems. Most of the proposals contain a similarity measure component, which returns the connection strength between two objects in the form of a number between 0 and 1. Pairs of entities having a value near to 1 probably represent the same real world concepts. Hence they must be connected during the schema or application integration task. On the other hand, entities described by lower values are out of interest from the point of view of integration. Similarity measurement methods can save a lot of human efforts this way. Thus preventing the necessity of the presence of a human expert during the schema matching task is also a reasonable goal by developing different approaches and methodologies.

There are 3 basic types of schema matching approaches today:
- Linguistic approaches examine the naming similarity of entities using different string comparing functions for example searching for sub-strings or concatenations. Usually they are also extended by (domain specific) dictionaries and taxonomies to be able to detect the similarities in the meaning of schema concepts as well.
- Structural methods are based on the comparing of paths connected to the given entities leading to the leaves, children or to the root element. The main idea behind this approach is, that two entities of two different schemas probably represent the same real world entity if their structural neighborhood is built similarly e.g. the two paths leading to the root element are similar. Path similarities are mainly measured by defining indicators for similar node correspondence, node order, etc.
- Combining the two approaches above and applying more specific algorithms within solid frameworks results in a solution called combined approach. Because of its robustness and effectiveness, most of the presented solutions are from this category in current literature. However algorithm complexity and computational costs are hardly taken into account.

There are many possible area of application e.g. aligning of service interfaces in a SOA based integration scenario, where schema matching algorithms should be
performed quite often during the everyday work. In this case, it is important to have an algorithm, which is able to be executed in an acceptable time. Therefore analyzes and prediction of algorithms’ runtime is at least as essential as the evaluation of the accuracy of their results.

This paper presents mathematical methods and techniques to predict computational costs and hence execution run-time of schema matching approaches. The structure of the paper is as follows: the next chapter summarizes works related to ours. In chapter 3 we present the problem of computational requirement estimation and describe our solution. Chapter 4 contains experimental results validating our method. Chapter 5 briefly describes algorithm accuracy and finally chapter 6 concludes the achieved results.

2 Related work

There are numerous researches in literature about schema matching [1, 2, 3, 5, 6, 8, 9, 10, 11]. A special approach, called similarity flooding is presented in the work [8], which is hard to be identified upon the classification of schema matching approaches introduced above. The main idea behind the similarity flooding is that the similarity of two given entities can be measured by the similarity of their neighborhoods. The paper also presents detailed analyzes and comparison of results to other approaches. However this is performed only from the point of view of accuracy and the paper lacks any kind of estimation of run-time costs. This algorithm is described in details and compared with my approach later in chapters 3 and 4.

The authors in [2] present a conventional example of a combined approach. The evaluation starts with a linguistic analyzes based on the open dictionary called WordNet[4], but the main added value is the comprehensive structural method performed on the schema trees. Unfortunately this paper also lacks of computational complexity estimation.

Although this requires the handling of functions working also on graphs, matrices and other ordered data structures, our solution is able to predict the computational requirement also for this approach – see later in chapters 3 and 4.

A generic schema matching tool called COMA++ is presented in [5]. It provides a library of individual matchers realizing a flexible platform for a combined matcher. The application of different matching strategies and the decomposition of large schemas with a fragment-based matcher into smaller sets ensure high scalability for the presented schema matching solution. The approach is also evaluated on schemas from various sizes (containing a number of nodes between 27 and 843 and a number of possible paths between 34 and 26228) and it is also compared with other proposals extensively. However the focus is mainly set on accuracy comparison while it a proper and detailed analyzes of run-time cost is still missing.

3 Algorithm complexity of approaches

Besides of the expected accuracy of the results the run-time cost of an algorithm is also a key aspect by finding a solution for the schema matching problem. More accurate solutions may require much more resources (e.g. computational performance) to provide results. This can lead to enormous long run-time in the given hardware configurations which can not be accepted by some system e.g. online, real time systems or even by a solid development environment.

The costs of an algorithm are strictly connected with the complexity of the given method. We have proved that complexity is proportional by the expected number of the steps at the execution. Because there is a (single) computer operation in the background of each performed step, the actual run-time of given computer configurations can also be predicted for given schemas this way. Hence the number of the expected steps is calculated by our method. In the next few paragraphs we present the main tasks and tools of our method. For better understanding tasks of real schema matching solutions are also described and evaluated.

3.1 Examples for schema matching approaches

The proposal similarity flooding is based on the following idea: if a graph is constructed where nodes represent entities like complex types or attributes and edges represent relationship among them like containing, inheritance or association, the relation between two given entity of the schemas is determined by the relation of their neighborhood. In other worlds, if the nodes close to entity A in the fist schema are semantically related to corresponding nodes close to entity B in the second schema, then entity A and entity B are probably semantically related and vice versa [2].

The key point of the approach is the algorithm for similarity flooding, where similarity of the nodes is distributed among the neighbors. This is performed for all nodes in a specially constructed pair wise connectivity graph (PCG). Furthermore this is an iterative step, which is stopped by reaching a stable state or a given number of ran iterative steps. The results (similarity values of entities creating common nodes in the PCG) are directly readable after flooding is stopped.

The next presented approach - called WordNet in our article- has a strong structural analyzer part but relies on the values of a dictionary based matcher, the WordNet [2, 4]. After the initial values of similarity between
The complexity in the next few sub-chapters.

and analyzed from the point of computational algorithms, see later.

paper – definitely outperforms other presented approaches. On the other hand its accuracy is also not lower so this approach – called NTA in the rest of the paper – definitely outperforms other presented algorithms, see later.

Every main steps of every approach will be presented and analyzed from the point of computational complexity in the next few sub-chapters.

### 3.2 Complexity of simple comparisons and constructional tasks

This section presents the computational complexity of some simpler tasks upcoming for example in the approach NTA.

By evaluating the linguistic correspondence between the naming of different complex types of schemas we use function \( N(C_i, G_j) \), which is a single one-step operation – function substring. So the number of steps of function \( N(C_i, G_j) \) is 1 for a given evaluation of two different complex types.

Calculating the similarity based on attached terms uses the function \( T(C_i, G_j) \). This compares all attached concepts of evaluated couples what results in the number of steps, where \( \#\text{Terms}(C_i) \) is the number of terms connected to complex type \( C_i \).

The correspondence between the attributes of complex types is calculated by the function \( A_i(C_i, G_j) \). Within this every attribute of the local schema is compared with every attribute of the global schema resulting number of steps \( \#\text{Attr}(C_i) \#\text{Attr}(G_j) \) where \( \#\text{Attr}(C_i) \) represents the number of attributes connected to complex type \( C_i \).

By the comparisons of two whole schemas all functions depicted above should be calculated to all pairs of the complex types of the schemas. This means \(#\text{Steps}_{\text{NTA, Perf}}(C,G) = \sum_{i,j} \#\text{Term}(C_i) \#\text{Term}(G_j) + \#\text{Attr}(C_i) \#\text{Attr}(G_j) \) steps for schemas \( C \) and \( G \).

These expressions should be applied for every simple comparing steps of every schema matching approaches e.g. the retrieving of the similarity values from the dictionary in proposal Wordnet or by the creation of initial values between the OIM graph nodes in the similarity flooding.

Most of the algorithm requires some preparation e.g. constructing trees from complex types before its execution. This requires the processing of all complex types with their all attributes which costs

\[
\text{Steps}_{\text{NTA, Prep}}(C,G) = \sum_i \#\text{Attr}(C_i) + \sum_j \#\text{Attr}(G_j) + \#\text{C} + \#\text{G} + \\
+ \#\text{Term}(C_i) + \#\text{Term}(G_j)
\]

where \( \#\text{C} \) is the number of the complex types in the first schema and \( \#\text{G} \) denotes the number of complex types in the second schema. So similar to the computational requirements of preparation in approach NTA, every preparation steps of every matching algorithms should be estimated with the expression above.

By the similarity flooding an open information model (OIM) based graph is constructed both from the local schemas. Similarly to initial graph construction by the NTA approach this step requires

\[
\text{Steps}_{\text{OIM}}(C,G) = 2 \left( \sum_i \#\text{Attr}(C_i) + \sum_j \#\text{Attr}(G_j) + \\
+ \#\text{C} + \#\text{G} + \#\text{Y}_C + \#\text{Y}_G + 3 \right)
\]

where \( \#\text{Y}_C \) (\( \#\text{Y}_G \)) is the number of existing data types in the first schema (second schema respectively) and the constant value of 3 is the insertion cost of the tree special category node (complex type, attribute, and attribute type) at the end of the expression.

The computational complexity of tree construction step of the approach Wordnet can also be predicted this way. So the first step of approach Wordnet requires

\[
\text{Steps}_{\text{TreeConn}}(C,G) = \sum_i \#\text{Attr}(C_i) + \sum_j \#\text{Attr}(G_j) + \#\text{C} + \#\text{G}
\]

computational steps.

The second task of approach Wordnet should be mentioned here as well. Semantic relevance of entities of both schemas is retrieved from the Wordnet database in a number of steps as follows:

\[
\text{Steps}_{\text{WordNet}}(C,G) = \left( \#\text{C} + \sum_i \#\text{Simple Attr}(C_i) \right) \cdot \\
\left( \#\text{G} + \sum_j \#\text{Simple Attr}(G_j) \right)
\]

### 3.3 Complexity of special graph traversals

An initial mapping between the nodes of the two OIM graphs is calculated in approach similarity flooding. This will represent a rough starting value for the similarity of nodes (used later by the initializing of the flooding algorithm at the PCG). The generated
number of nodes to each complex type is 2 (1 for the entity and 1 for its name). Similarly, all attributes and possible range types will also be presented by 2 nodes in the OIM. Hence the number of generated nodes is 2 times the number of complex types plus 2 times the number of attributes plus 2 times the number of data types existing in the given schema.

Besides, there are exactly 2 nodes generated to every complex type and 3 initial nodes for signing complex type, attribute and data type category nodes. During the creation of initial mapping every node of the 2 generated OIM schemas must be compared to every node of the other schema. Thus the number of the steps (comparisons) performing in this section is the following:

\[ \text{Steps}_{\text{init-map}}(C, G) = \left(3 + 2|\mathcal{C}| + 2 \sum \text{Attr}(C_i) \right) \ast \left(3 + 2|\mathcal{G}| + 2 \sum \text{Attr}(G_j) \right) \]

(6)

So the required steps of comparisons were calculated by determining the nodes created in each steps of the graph construction. This kind of deductions can also be used by analyzing of various matchers containing special graph-based structures.

Another typical example for this could be the evaluation of the PCG graph creation step by the similarity flooding approach. The pair wise connectivity graph (PCG) is created along the same type of edges of the two schemas. (The type of edges can be type, name, and other relation like attribute and data type). To do so, every edge of the same type from the two schemas is compared. Thus the number of the steps of the similarity flooding approach is as follows:

\[ \text{Steps}_{\text{PCGFlooding}}(C, G) = \sum \left(3 + 2|\mathcal{C}| + 2 \sum \text{Attr}(C_i) \right) \ast \left(3 + 2|\mathcal{G}| + 2 \sum \text{Attr}(G_j) \right) \]

(7)

To the estimation of the steps of the flooding itself we need to know the exact number of the nodes in the PCG graph. A deduction similarly to the above presented cases leads to expression (8):

\[ \text{Nodes}_{\text{PCG}}(C, G) = \sum \left(3 + 2|\mathcal{C}| + 2 \sum \text{Attr}(C_i) \right) \ast \left(3 + 2|\mathcal{G}| + 2 \sum \text{Attr}(G_j) \right) \]

Thus the flooding itself is performed on each node of the PCG the required number of its computational steps is as follows:

\[ \text{Steps}_{\text{Flooding}}(C, G) = \sum_{\text{steps}} \text{Nodes}_{\text{PCG}}(C, G) \]

(9)

\[ \sum_{\text{steps}} \left(3 + 2|\mathcal{C}| + 2 \sum \text{Attr}(C_i) \right) \ast \left(3 + 2|\mathcal{G}| + 2 \sum \text{Attr}(G_j) \right) \]

where \( \sum \) signs that this must be performed for all iterative steps.

### 3.4 Complexity of structure comparisons

The structural similarity of graph nodes is calculated by 3 aspects by the approach Wordnet. The ancestor, child and leaf context all contribute to the final value of the similarity. The ancestor similarity is calculated as follows:

\[ \text{Sim}_{\text{ancestor}}(C_i, G_j) = \text{PS}(P_i, P_j) \ast \text{LS}(C_i, G_j) \]

(10)

where \( \text{PS}(P_i, P_j) \) is the path similarity function for the path starting from entities \( C_i, G_j \) respectively to the root element of the given trees and function \( \text{LS}(C_i, G_j) \) represents the result of WordNet request for \( C_i \) and \( G_j \). Because the values of LS are already available only the two paths should be found. Because the computational complexity of the functions contained by the path similarity are in order \( O(n^2) \), the a number of steps in this task is estimated as follows (11):

\[ \text{Steps}_{\text{ancest}}(C_i, G_j) = \text{length}(P_i) \ast \text{length}(P_j) \]

where function \( \text{length}() \) returns the number of nodes in a given path.

For the overall method of the calculation of ancestor context this must be performed for all possible pairs of complex types of the two schemas. However we face a serious problem by evaluating the expression above for real schemas: the length of the path is different value for every node of the tree. To be able to solve the problem, the length of paths within a schema is estimated by a constant value. For this we have chosen the average lengths to the root calculated as follows:

\[ \text{AVG}_{\text{length}}(C_i) = \frac{1}{|\mathcal{C}|} \sum \text{length}(P_i) \]

(12)

Using this value within the expression above the overall computational cost of the ancestor context calculation can already be predicted.

Based on our experiments we claim that this approximation method be a good estimation for all kind of algorithms where path analyzing (comparing) functions are used.

For example it is also applicable by the calculation of the leaf similarity. Due to the specification of the Wordnet approach, the path similarity (PS) must be calculated to all paths leading to (descendants) leaves of the given pair of concepts originating from the 2 different schemas. The computational cost of the evaluation of two entities is as follows:

\[ \text{Steps}_{\text{leaf-path}}(C_i, G_j) = \sum \text{length}(P_i) \ast \text{length}(P_j) \]

(13)

where similarly to the ancestor calculation the length is estimated with an average value of paths to the leaves.

The child context is also based on the function LS. However it requires no path similarity functions because only the set of associated children nodes are compared in this step.
3.5 Computational complexity of sorting lists

There are many situations where a simple sorting of a list of values is required by a task of a schema matching algorithm. The algorithm theory has all the necessary results which should be adopted here. However we should not forget about the exact type of applied ordering method. In our implementation we used a kind of quick-sort algorithm whose computational cost can be estimated by the following average value:

\[
\text{Steps}_{\text{quick-sort}} = n \cdot 1.39 \cdot \log_2 n
\]

By the calculation of the exact value of child and leaf similarities only the highest 25% (or 50%) of the above presented functions must be taken into account. This requires the sorting of the results, which is

\[
\text{Steps}_{\text{sortingChild}}(C_i, G_j) \approx \left( \text{child}(C_i) \cdot \text{child}(G_j) \right) \cdot 1.39 \cdot \log_2 \left( \text{child}(C_i) \cdot \text{child}(G_j) \right)
\]

for the child context and

\[
\text{Steps}_{\text{sortingLeaf}}(C_i, G_j) \approx \left( \text{leaf}(C_i) \cdot \text{leaf}(G_j) \right) \cdot 1.39 \cdot \log_2 \left( \text{leaf}(C_i) \cdot \text{leaf}(G_j) \right)
\]

for the leaf context.

3.6 Number of expected steps of the approaches

In this paragraph we present the expected number of steps of each analyzed approach in a complex form without detailed explanation. Deductions and detailed expressions can be constructed based on the specification of the approaches and the expressions and methods presented above.

\[
\text{Steps}_{\text{NTA}}(C, G) = \text{Steps}_{\text{NTA,Petion}}(C, G) + \text{Steps}_{\text{NTA,Prop}}(C, G)
\]

\[
\text{Steps}_{\text{Sim,Flood}}(C, G) = \text{Steps}_{\text{OM}}(C, G) + \text{Steps}_{\text{sim,rep}}(C, G) + \text{Steps}_{\text{sim,flood}}(C, G)
\]

\[
\text{Steps}_{\text{combined}}(C, G) = \text{Steps}_{\text{TreeCount}}(C, G) + \text{Steps}_{\text{WordNet}}(C, G) + \sum_{i,j} \text{Steps}_{\text{sim}}(C_i, G_j) + \text{Steps}_{\text{step}}(C_i, G_j) + \text{Steps}_{\text{sim}}(C_i, G_j)
\]

In the next section the calculation complexity of the different approaches is evaluated on multiple test cases.

4 Experiments

To estimate and analyze working cost i.e. number of run-time steps for different schemas more samples was implemented. As the reader could see in the previous chapter, calculation of the expected number of run-time steps is not an easy method especially for custom (e.g. real-life) scenarios. Thus specific samples were created so that the required parameters e.g. average path length to the root, or to the leaves can be calculated. The implemented test schemas have the following parameters, see table 1.

<table>
<thead>
<tr>
<th>Brances</th>
<th>Depthness</th>
<th>Attributes</th>
<th>Terms</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sample</td>
<td>1.33</td>
<td>1.375</td>
<td>5</td>
</tr>
<tr>
<td>Example1</td>
<td>2</td>
<td>2</td>
<td>7</td>
</tr>
<tr>
<td>Example2</td>
<td>3</td>
<td>4</td>
<td>9</td>
</tr>
<tr>
<td>Example3</td>
<td>5</td>
<td>5</td>
<td>12</td>
</tr>
</tbody>
</table>

To be able to calculate the number of expected steps the following values were also approximated:

- number of different simple types and flooding iterations for approach similarity flooding and
- average number of children and leaves, and average length of path to the root and to the leaves for calculating the steps of the combined matcher (approach WordNet).

The test cases were defined as all possible combination of schema pairs. Including also the evaluation of the same schemas this means 10 test-cases from different complexity. The estimated number of run-time steps for every approach is presented in table 2.

<table>
<thead>
<tr>
<th>Típus</th>
<th>NTA</th>
<th>Sim. Flood</th>
<th>WordNet</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sample vs Sample</td>
<td>1375</td>
<td>19801</td>
<td>56981</td>
</tr>
<tr>
<td>Example1 vs Sample</td>
<td>2451</td>
<td>33663</td>
<td>95619</td>
</tr>
<tr>
<td>Example1 vs Example1</td>
<td>4396</td>
<td>57335</td>
<td>252207</td>
</tr>
<tr>
<td>Example2 vs Example2</td>
<td>59471</td>
<td>495365</td>
<td>5865024</td>
</tr>
<tr>
<td>Example2 vs Example1</td>
<td>91809</td>
<td>847019</td>
<td>120860636</td>
</tr>
<tr>
<td>Example3 vs Example2</td>
<td>1921843</td>
<td>16930913</td>
<td>610377358</td>
</tr>
<tr>
<td>Example3 vs Example3</td>
<td>2152256</td>
<td>20763658</td>
<td>361249806</td>
</tr>
<tr>
<td>Example3 vs Sample</td>
<td>3882655</td>
<td>35530560</td>
<td>609118756</td>
</tr>
<tr>
<td>Example3 vs Example3</td>
<td>81375643</td>
<td>1204098630</td>
<td>37622299563</td>
</tr>
<tr>
<td>Example3 vs Example3</td>
<td>3448208988</td>
<td>50469880392</td>
<td>2.31949E+12</td>
</tr>
</tbody>
</table>

To validate the correctness of our approximation method specific step counters were placed into the implementation of the algorithms. The number of steps measured during the execution is shown in table 3.

<table>
<thead>
<tr>
<th>Típus</th>
<th>NTA</th>
<th>Sim. Flood</th>
<th>WordNet</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sample vs Sample</td>
<td>1375</td>
<td>18827</td>
<td>54256</td>
</tr>
<tr>
<td>Example1 vs Sample</td>
<td>2451</td>
<td>32837</td>
<td>113933</td>
</tr>
<tr>
<td>Example1 vs Example1</td>
<td>4369</td>
<td>57371</td>
<td>240700</td>
</tr>
<tr>
<td>Example2 vs Sample</td>
<td>59471</td>
<td>483215</td>
<td>582161</td>
</tr>
<tr>
<td>Example1 vs Example2</td>
<td>91809</td>
<td>847037</td>
<td>12024888</td>
</tr>
<tr>
<td>Example2 vs Example2</td>
<td>1921843</td>
<td>16930931</td>
<td>622136404</td>
</tr>
<tr>
<td>Example3 vs Example2</td>
<td>2152256</td>
<td>20255808</td>
<td>380147160</td>
</tr>
<tr>
<td>Example3 vs Example3</td>
<td>3882655</td>
<td>65530578</td>
<td>790027687</td>
</tr>
<tr>
<td>Example3 vs Example3</td>
<td>81375643</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>Example3 vs Example3</td>
<td>3448208988</td>
<td>N/A</td>
<td>N/A</td>
</tr>
</tbody>
</table>

The measured values have successfully validated our approximation method. Not surprisingly by the approach NTA the same values were returned by the test, because neither average calculation nor any other approximations were applied there. But the measured values of other approaches are also close to our estimations. The “N/A” symbol stands for not successful execution of the given test scenarios. Because of the size of these scenarios and the higher complexity of algorithms they can not be
executed on regular hardware configurations e.g. on personal computers or the run-time is enormous long.

Based on our calculations the order of the actual execution is also predictable. However without knowing the exact capacity of run-time configuration and execution of some simpler scenarios for achieving some reference measurements more accurate estimation of execution time is not possible. On the other hand, fully accurate run times are not need in most of the cases, only the correct order of magnitude is important. Our measured execution times are shown in table 4.

### Table 4 Measured execution times of the tests (seconds)

<table>
<thead>
<tr>
<th>Types</th>
<th>NTA</th>
<th>Sim. Flood</th>
<th>WordNet</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sample vs Sample</td>
<td>0,03</td>
<td>0,06</td>
<td>0,08</td>
</tr>
<tr>
<td>Example1 vs Example1</td>
<td>0,03</td>
<td>0,08</td>
<td>0,11</td>
</tr>
<tr>
<td>Example2 vs Sample</td>
<td>0,08</td>
<td>0,63</td>
<td>1,00</td>
</tr>
<tr>
<td>Example1 vs Example2</td>
<td>0,08</td>
<td>1,11</td>
<td>1,95</td>
</tr>
<tr>
<td>Example2 vs Example2</td>
<td>0,92</td>
<td>25,94</td>
<td>98,23</td>
</tr>
<tr>
<td>Example3 vs Example3</td>
<td>3,56</td>
<td>41,91</td>
<td>91,88</td>
</tr>
<tr>
<td>Example1 vs Example3</td>
<td>2,52</td>
<td>206,48</td>
<td>159,06</td>
</tr>
<tr>
<td>Example2 vs Example3</td>
<td>40,52</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>Example3 vs Example3</td>
<td>2264,84</td>
<td>N/A</td>
<td>N/A</td>
</tr>
</tbody>
</table>

### 5 Short evaluation of accuracy

As already mentioned above by evaluating schema matching approaches both accuracy and required computational cost should be taken into account. Although the aim of this paper was a detailed analyzes of run-time cost the results of our accuracy measurements are also presented in table 5. As measurement methods and units the mostly used parameters were adapted from current literature [2, 8].

<table>
<thead>
<tr>
<th>Scenarios:</th>
<th>Scenario 1</th>
<th>Scenario 2</th>
<th>Scenario 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Indicators</td>
<td>NTA</td>
<td>SF</td>
<td>WN</td>
</tr>
<tr>
<td>Precision</td>
<td>0,8</td>
<td>0,25</td>
<td>1</td>
</tr>
<tr>
<td>Recall</td>
<td>1</td>
<td>0,75</td>
<td>0,46</td>
</tr>
<tr>
<td>F-Measure</td>
<td>0,89</td>
<td>0,86</td>
<td>0,43</td>
</tr>
</tbody>
</table>

Our previously developed schema matching algorithm (called NTA) performed quite well comparing to others. Taking also into account its much lower computational requirement it clearly outperforms other approaches.

### 6 Conclusions

The computational cost of schema matching approaches were analyzed and estimated by different mathematical methods and techniques. This is useful by designing and comparing of schema matching solutions willing to be used in everyday work or at critical on-line systems. The presented approach is applicable for various kinds of matching approaches; typical computational tasks of linguistic, structural and combined matchers were all covered. As a demonstration, we have also successfully validated our method on three different approaches. Our experiments showed that the estimated values are correct, and the required run-time complexity and the order of magnitude of the actual execution times are also predictable based on our returned results.

### References:


