QOM - Quick Ontology Mapping

Marc Ehrig and Steffen Staab

Institute AIFB, University of Karlsruhe

Abstract. (Semi-)automatic mapping — also called (semi-)automatic alignment — of ontologies is a core task to achieve interoperability when two agents or services use different ontologies. In the existing literature, the focus has so far been on improving the quality of mapping results. We here consider QOM, Quick Ontology Mapping, as a way to trade off between effectiveness (i.e. quality) and efficiency of the mapping generation algorithms. We show that QOM has lower run-time complexity than existing prominent approaches. Then, we show in experiments that this theoretical investigation translates into practical benefits. While QOM gives up some of the possibilities for producing high-quality results in favor of efficiency, our experiments show that this loss of quality is marginal.

1 Introduction

Semantic mapping[1] between ontologies is a necessary precondition to establish interoperability between agents or services using different ontologies.

The problem of finding adequate alignments is a hard one. In fact, it is hard with regard to several dimensions, two of which we focus on in this paper: 1. Effectiveness: We demand high-quality alignments for ontology schemas and metadata. 2. Efficiency: The size of ontologies tends to grow and ontologies with several thousand entities become commonplace. Even finding a proposal for an alignment of ontology subsets becomes a problem of practical tractability.

In recent years we have seen a range of research work on methods proposing such mappings [AS01,NM03,DDH03]. The focus of the previous work, however, has been laid exclusively on improving the *effectiveness* of the approach (i.e. the quality of proposed mappings such as evaluated against some human judgement given either a posteriori or a priori). When we tried to apply these methods to some of the real-world scenarios we address in other research contributions [EHvH+03], we found that existing mapping methods were not suitable for the ontology integration task at hand, as they all neglected *efficiency*.

To illustrate our requirements: We have been working in realms where light-weight ontologies are applied such as the ACM Topic hierarchy with its 10^4 concepts or folder structures of individual computers, which corresponded to 10^4 to 10^5 concepts. Finally, we are working with Wordnet exploiting its 10^6 concepts (cf. [HSS03]) or 10^7 concepts in UMLS. When mapping between such light-weight ontologies, the trade-off that one has to face is between effectiveness and efficiency. For instance, consider the knowledge management platform built on a Semantic Web And Peer-to-peer basis in SWAP

[1] Frequently also called alignment.

[EHvH+03]. It is not sufficient to provide its user with the best possible mapping, it is also necessary to answer his queries within a few seconds — even if two peers use two different ontologies and have never encountered each other before.

In this paper we present an approach that considers both the quality of mapping results as well as the run-time complexity. Our hypothesis is that mapping algorithms may be streamlined such that the loss of quality (compared to a standard baseline) is marginal, but the improvement of efficiency is so tremendous that it allows for the ad-hoc mapping of large-size, light-weight ontologies. To substantiate the hypothesis, we outline a comparison of the worst-case run-time behavior (given in full detail in [ES04a]) and we report on a number of practical experiments. The approaches used for our comparison represent different classes of algorithms for ontology mapping. Comparing to these approaches we can observe that our new efficient approach QOM achieves good quality. The complexity of QOM is of $O(n \cdot log(n))$ (measuring with n being the number of the entities in the ontologies) against $O(n^2)$ for approaches that have similar effective outcomes.

The remainder of the paper starts with a clarification of terminology (Section 2). To compare the worst-case run-time behavior of different approaches, we first describe a canonical process for ontology mapping that subsumes the different approaches compared in this paper (Section 3). The process is a core building block for later deriving the run-time complexity of the different mapping algorithms. Section 4 further presents basic tools for these algorithms. In Section 5, different approaches for proposing mappings are described and aligned to the canonical process. The way to derive their run-time complexity is outlined in Section 6. Experimental results (Section 7) complement the run-time considerations of the comparison.

2 Terminology

As a first step we want to present the basic definitions we require for our mapping process and algorithms.

2.1 Ontology

The underlying data models in our process are ontologies. To facilitate the further description, we briefly summarize their major primitives and introduce some shorthand notations. In the understanding of this paper it consists of both schema and metadata. As we currently focus on light-weight ontologies, we build on RDF/S[2] to represent ontologies. To facilitate the further description, we briefly summarize its major primitives and introduce some shorthand notations. In the understanding of this paper it consists of both schema and metadata. An RDF model is described by a set of statements, each consisting of a subject, a predicate and an object.

$$O := (C, H_C, R_C, H_R, I, R_I, A)$$

[2] http://www.w3.org/RDFS/

An ontology O is defined by its set of Concepts $\mathcal{C}$ (instances of "rdfs:Class") with a corresponding subsumption hierarchy H_C (a binary relation corresponding to "rdfs:subClassOf"). Relations $\mathcal{R}$ (instances of "rdf:Property") exist between single concepts. Relations are arranged alike in a hierarchy H_R ("rdfs:subPropertyOf"). An entity $i \in \mathcal{I}$ may be an instance of a class $c \in \mathcal{C}$ ("rdf:type"). An instance $i \in \mathcal{I}$ may have one j or many role fillers from $\mathcal{I}$ for a relation r from $\mathcal{R}$. We also call this type of triple (i, r, j) a property instance. Additionally one can define axioms A which can be used to infer knowledge from already existing one. An extended definition of ontologies can be found in [Gru93] and [SEH+03].

```
<rdf:RDF>
<rdfs:Class rdf:ID="Vehicle"/>
<rdfs:Classrdf:ID="Car">
    <rdfs:subClassOf rdf:resource="#Vehicle"/>
</rdfs:Class>
<rdf:Property rdf:ID="hasSpeed">
    <rdfs:domain rdf:resource="#Car"/>
    <rdfs:range rdf:resource="#Speed"/>
</rdf:Property>
<example:Speed rdf:ID="250 km/h"/>
<example:Car rdf:ID="Porsche KA-123">
    <example:hasSpeed rdf:resource="#250 km/h">
</exmaple:Car>
</rdf:RDF>
```

Example 1. An ontology describing a specific car.

OWL[3] is another language offering more complex modelling primitives.

2.2 Mapping

Goal of the process is to provide correct mappings. Due to the wide range of expressions used in this area (merging, alignment, integration etc.), we want to describe our understanding of the term "mapping": Given two ontologies O_1 and O_2, mapping one ontology onto another means that for each entity (concept C, relation R, or instance I) in ontology O_1, we try to find a corresponding entity, which has the same intended meaning, in ontology O_2. [Su02] set a pointer to this definition.

Definition 1. *We define an ontology mapping function,* map*, based on the vocabulary,* $\mathcal{E}$*, of all terms* $e \in \mathcal{E}$ *and based on the set of possible ontologies,* $\mathcal{O}$*, as a partial function:*

$$\mathrm{map} : \mathcal{E} \times \mathcal{O} \times \mathcal{O} \rightharpoonup \mathcal{E},$$

$$\textit{with } \forall e \in O_1(\exists f \in O_2 : \mathrm{map}(e, O_1, O_2) = f \vee \mathrm{map}(e, O_1, O_2) = \perp).$$

A term e interpreted in an ontology O is either a concept, a relation or an instance, i.e. $e_{|O} \in \mathcal{C} \cup \mathcal{R} \cup \mathcal{I}$. We usually write e instead of $e_{|O}$ when the ontology O is clear from the context of the writing. We write $\mathrm{map}_{O_1,O_2}(e)$ for $\mathrm{map}(e, O_1, O_2)$. We derive

[3] http://www.w3.org/OWL/

a relation map_{O_1,O_2} by defining $\text{map}_{O_1,O_2}(e,f) \Leftrightarrow \text{map}_{O_1,O_2}(e) = f$. We leave out O_1, O_2 when they are evident from the context and write $\text{map}(e) = f$ and $\text{map}(e,f)$, respectively. Once a (partial) mapping, map, between two ontologies O_1 and O_2 is established, we also say "*entity e is mapped onto entity f*" iff $\text{map}(e,f)$. An entity can either be mapped to at most one other entity. A pair of entities (e,f) that is not yet in map and for which appropriate mapping criteria still need to be tested is called a *candidate mapping*.

It is the purpose of this paper to define effective and efficient mechanisms to define map_{O_1,O_2}. However, we do not consider its further use, e.g. for query answering over different ontologies and the different ways that this can be done, such as immediate translation of data (cf. [MMSV02]) or theory approximation (cf. [Stu02]).

2.3 Example

The following example illustrates a mapping. Two ontologies O_1 and O_2 describing the domain of car retailing are given (Figure 1). A reasonable mapping between the two ontologies is given in Table 1 as well as by the dashed lines in the figure.

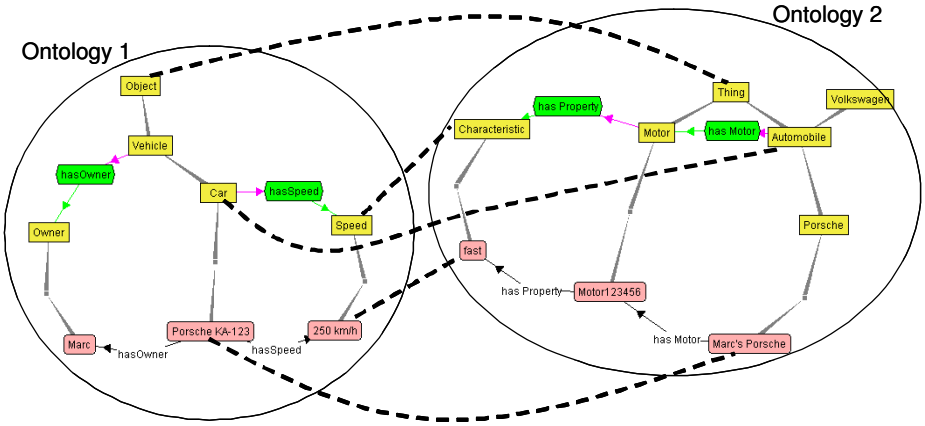

Fig. 1. Example Ontologies and their Mappings

Apart from one-to-one mappings as investigated in this paper one entity often has to be mapped to a complex composite such as a concatenation of terms (first and last name) or an entity with restrictions (a sports-car is a car going faster than 250 km/h). [DR02,DLD+04] propose approaches for this. We do not deal with such issues of complete ontology mappings here.

Ontology O_1	Ontology O_2
Object	Thing
Car	Automobile
Porsche KA-123	Marc's Porsche
Speed	Characteristic
250 km/h	fast

Table 1. Mapping Table for Relation $\mathrm{map}_{O_1,O_2}(e,f)$

3 Process

We briefly introduce a canonical process that subsumes all the mapping approaches we are aware of.[4] Figure 2 illustrates its six main steps. It is started with two ontologies, which are going to be mapped onto one another, as its input:

1. *Feature Engineering* transforms the initial representation of ontologies into a format digestible for the similarity calculations. For instance, the subsequent mapping process may only work on a subset of RDFS primitives. This step may also involve complex transformations, e.g. it may require the learning of classifiers as input to the next steps.
2. *Selection of Next Search Steps.* The derivation of ontology mappings takes place in a search space of candidate mappings. This step may choose, to compute the similarity of a restricted subset of candidate concepts pairs $\{(e,f)|e \in O_1, f \in O_2\}$ and to ignore others.
3. *Similarity Computation* determines similarity values between candidate mappings (e,f) based on their definitions in O_1 and O_2, respectively.
4. *Similarity Aggregation.* In general, there may be several similarity values for a candidate pair of entities e, f from two ontologies O_1, O_2, e.g. one for the similarity of their labels and one for the similarity of their relationship to other terms. These different similarity values for one candidate pair must be aggregated into a single aggregated similarity value.
5. *Interpretation* uses the individual or aggregated similarity values to derive mappings between entities from O_1 and O_2. Some mechanisms here are, to use thresholds for similarity mappings [NM03], to perform relaxation labelling [DDH03], or to combine structural and similarity criteria.
6. *Iteration.* Several algorithms perform an iteration over the whole process in order to bootstrap the amount of structural knowledge. Iteration may stop when no new mappings are proposed.
   Note that in a subsequent iteration one or several of steps 1 through 5 may be skipped, because all features might already be available in the appropriate format or because some similarity computation might only be required in the first round.

Eventually, the output returned is a mapping table representing the relation map_{O_1,O_2}.

[4] The process is inspired by CRISP-DM, http://www.crisp-dm.org/, the CRoss Industry Standard Process for Data Mining [She00].

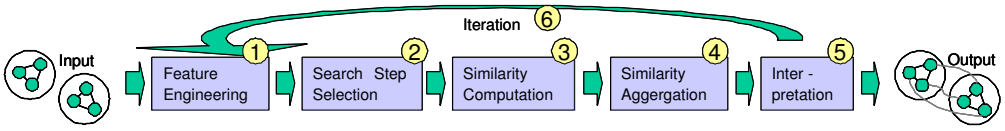

Fig. 2. Mapping Process

In the next sections we will further describe the concepts of the different steps. After that we model some of the best-known mapping algorithms according to this process in order to compare them against our own proposal QOM with regard to effectiveness as well as efficiency.

4 A Toolbox of Data Structures and Methods

The principal idea of this section is to provide a toolbox of data structures and methods common to many approaches that determine mappings. This gives us a least common denominator based on which concrete approaches instantiating the process depicted in Figure 2 can be compared more easily. The following Section 5 will then describe how existing mechanisms use this toolbox.

4.1 Features of Ontological Entities

To compare two entities from two different ontologies, one considers their characteristics, i.e. their features. We will now emphasize those features which can give hints on whether two entities are actually the same. Basically our assumption is that entities with the same features are identical. The features may be specific for a mapping generation algorithm, in any case the features of ontological entities (of concepts, relations, instances) need to be extracted from extensional and intensional ontology definitions. These features have to be determined by an expert understanding the encoded knowledge in ontologies. See also [ES04b] and [EV03] for an overview of possible features and a classification of them. Possible characteristics include:

- *Identifiers*: i.e. strings with dedicated formats, such as unified resource identifiers (URIs) or RDF labels.
- *RDF/S Primitives*: such as properties or subclass relations
- *Derived Features*: which constrain or extend simple RDFS primitives (e.g. most-specific-class-of-instance)
- *Aggregated Features*: i.e. aggregating more than one simple RDFS primitive, e.g. a sibling is every instance-of the parent-concept of an instance
- *OWL Primitives*: such as an entity being the sameAs another entity
- *Domain Specific Features* are features which only apply to a certain domain with a predefined shared ontology. For instance, in an application where files are represented as instances and the relation hashcode-of-file is defined, we use this feature to compare representations of concrete files.

Three are now exemplarily explained to illustrate the usage of features for mapping: *Labels* are human identifiers (names) for entities, normally shared by a community speaking a common language. We can therefore infer that if labels are the same, the entities are probably also the same. Concepts are arranged in a taxonomy. A resulting rule would be: if *sub-concepts* are the same, the actual concepts are also the same. The closer the common sub-concepts are to the compared concepts, the more information about similarity can be inferred[MMSV02]. Whereas the former two similarity hints were based on intensional features of ontologies, it is also possible to use extensional knowledge. Concepts are generalizations of *instances*. Therefore, if two concepts consist of the same instances they are the same.

Example We again refer to the example in Figure 1. The actual feature consists of a juxtaposition of relation name and entity name. The Car concept of ontology 1 is characterized through its (label, Car), the concept which it is linked to through (subclassOf, Vehicle), its (concept sibling, boat), and the (direct property, hasSpeed). Car is also described by its instances through (instance, Porsche KA-123). The relation hasSpeed on the other hand is described through the (domain, Car) and the (range, Speed). An instance would be Porsche KA-123, which is characterized through the instantiated (property instance, (hasOwner, Marc)) and (property instance, (hasSpeed, 250 km/h)).

4.2 Similarity Computation

Definition 2. *We define a similarity measure for comparison of ontology entities as a function as follows (cf. [Bis95]):*

$$\mathrm{sim} : \mathcal{E} \times \mathcal{E} \times \mathcal{O} \times \mathcal{O} \rightarrow [0, 1]$$

- $sim(e, f) = 1 \Leftrightarrow e = f$*: two objects are assumed to be identical.*
- $sim(e, f) = 0 \Leftrightarrow e \neq f$*: two objects are assumed to be different and have no common characteristics.*
- $sim(e, e) = 1$*: similarity is reflexive.*
- $sim(e, f) = sim(f, e)$*: similarity is symmetric.*[5]
- *Similarity and distance are inverse to each other.*

Different similarity measures $\mathrm{sim}_k(e, f, O_1, O_2)$ are indexed through a label k. Further, we leave out O_1, O_2 when they are evident from the context and write $\mathrm{sim}_k(e, f)$. The paper focuses on the similarity of pairs of single entities from different ontologies. At the current stage we do not compare whole ontologies or parts larger than one entity.

Similarity Measures The following similarity measures are needed to compare the features of ontological entities at iteration t.

[5] We assume symmetry in this paper, although we are aware that it is controversially discussed [MW01].

– *Object Equality* is based on existing logical assertions — especially assertions from previous iterations:

$$\mathrm{sim}_{obj}(a,b) := \begin{cases} 1 & map_{t-1}(a) = b, \\ 0 & otherwise \end{cases}$$

– *Explicit Equality* checks whether a logical assertion already forces two entities to be equal:

$$\mathrm{sim}_{exp}(a,b) := \begin{cases} 1 & \exists statement(a, \text{“}sameAs'', b), \\ 0 & otherwise \end{cases}$$

– *String Equality* is a strict measure to compare strings. All characters ($char(x)$ at position x) of the two strings have to be identical.

$$\mathrm{sim}_{strequ}(c,d) := \begin{cases} 1 & c.char(i) = d.char(i) \forall i \in [0,|c|] with |c| = |d| \\ 0 & otherwise \end{cases}$$

– *String Similarity* measures the similarity of two strings on a scale from 0 to 1 (cf. [MS02]) based on Levenshtein's edit distance, ed [Lev66].

$$\mathrm{sim}_{strsim}(c,d) := max(0, \frac{min(|c|,|d|) - ed(c,d)}{min(|c|,|d|)})$$

– *Dice Coefficient* compares two sets of entities [CAFP98].

$$\mathrm{sim}_{dice}(E,F) := \frac{|x \in (E \cap F)|}{|x \in (E \cup F)|}$$

– *SimSet*: For many features we have to determine to what extent two sets of entities are similar. As the individual entities have various and very different features, it is difficult to create a vector representing a whole sets of individuals. To remedy the problem, we use a technique known from statistics as multidimensional scaling [CC94]. We describe each entity through a vector representing the similarity to any other entity contained in the two sets. Multidimensional scaling assumes that if they have very similar distances to all other entities, they must be very similar. This is easily achieved, as we rely on other measures which already did the computation of similarity values $[0..1]$ between single entities. For both sets a representative vector is now created by determining an average vector over all individuals. Finally we determine the cosine between the two set vectors through the scalar product as the similarity value.

$$\mathrm{sim}_{set}(E,F) = \frac{\sum_{e \in E} \boldsymbol{e}}{|E|} \cdot \frac{\sum_{f \in F} \boldsymbol{f}}{|F|}$$

with $\boldsymbol{e} = (\mathrm{sim}(e,e_1), \mathrm{sim}(e,e_2), \ldots, \mathrm{sim}(e,f_1), \mathrm{sim}(e,f_2), \ldots)$, $\boldsymbol{f}$ analogously.

This list is by no means complete, but together with the features already allows to create very complex mapping measures, which are sufficient to describe most ontology mapping tools.

4.3 Similarity Aggregation

These measures are all input to the similarity aggregation. Similarities are aggregated by:

$$\mathrm{sim}_{agg}(e,f) = \frac{\sum_{k=1\ldots n} w_k \cdot \mathrm{adj}(\mathrm{sim}_k(e,f))}{\sum_{k=1\ldots n} w_k}$$

with w_k being the weight for each individual similarity measure, and adj being a function to transform the original similarity value ($adj : [0,1] \rightarrow [0,1]$), which yields better results.

One of the following measures is each applied with an ontology feature as described in the beginning of this section. Some features require syntactic comparisons, some object comparisons, and yet some others set comparisons. This list could be continued, in general, but is sufficient for our subsequent analysis.

4.4 Interpretation

From the similarity values we derive the actual mappings. The basic idea is that each entity may only participate in one mapping and that we assign mappings based on a threshold t and a greedy strategy that starts with the largest similarity values first. Ties are broken arbitrarily by $\tilde{argmax}_{(g,h)}$, but with a deterministic strategy.

$P(\bot, \bot, E \cup \{\bot\}, E \cup \{\bot\})$.
$P(g,h,U\backslash\{e\}, V\backslash\{f\}) \leftarrow P(e,f,U,V) \wedge \mathrm{sim}(g,h) > t$
$\quad \wedge (g,h) = \tilde{\mathrm{argmax}}_{(g,h)\in U\backslash\{e\}\times V\backslash\{f\}} \mathrm{sim}_{\mathrm{agg}}(g,h)$.
$\mathrm{map}(e,f) \leftarrow \exists X_1, X_2 P(e,f,X_1,X_2) \wedge (e,f) \neq (\bot,\bot)$.

5 Approaches to Determine Mappings

In the following we now use the toolbox, and extend it, too, in order to define a range of different mapping generation approaches. In the course of this section we present our novel Quick Ontology Mapping approach — QOM.

5.1 NOM - Naive Ontology Mapping

Our Naive Ontology Mapping (NOM)[ES04b] constitutes a straight forward baseline for later comparisons. It is defined by the steps of the process model as follows.

1. Feature Engineering Firstly, the ontologies for the NOM approach have to be in a format equivalent to the RDFS format. We use features as shown in Section 4.1.

2. Search Step Selection The selection of data for the comparison process is straight forward. All entities of the first ontology are compared with all entities of the second ontology. Any pair is treated as a candidate mapping.

3. Similarity Computation The similarity computation between an entity of O_1 and an entity of O_2 is done by using a wide range of similarity functions. Each similarity

Comparing	No.	Feature	Similarity Measure
Concepts	1	(label,X_1)	string similarity(X_1, X_2)
	2	(URI_1)	string equality(URI_1, URI_2)
	3	(X_1,sameAs,X_2) relation	explicit equality(X_1, X_2)
	4	(direct properties,Y_1)	SimSet(Y_1, Y_2)
	5	all (inherited properties,Y_1)	SimSet(Y_1, Y_2)
	6	all (super-concepts,Y_1)	SimSet(Y_1, Y_2)
	7	all (sub-concepts,Y_1)	SimSet(Y_1, Y_2)
	8	(concept siblings,Y_1)	SimSet(Y_1, Y_2)
	9	(direct instances,Y_1)	SimSet(Y_1, Y_2)
	10	(instances,Y_1)	SimSet(Y_1, Y_2)
Relations	1	(label,X_1)	string similarity(X_1, X_2)
	2	(URI_1)	string equality(URI_1, URI_2)
	3	(X_1,sameAs,X_2) relation	explicit equality(X_1, X_2)
	4	(domain,X_d1) and (range,X_r1)	object equality(X_{d1}, X_{d2}), (X_{r1}, X_{r2})
	5	all (super-properties,Y_1)	SimSet(Y_1, Y_2)
	6	all (sub-properties,Y_1)	SimSet(Y_1, Y_2)
	7	(property siblings,Y_1)	SimSet(Y_1, Y_2)
	8	(property instances,Y_1)	SimSet(Y_1, Y_2)
Instances	1	(label,X_1)	string similarity(X_1, X_2)
	2	(URI_1)	string equality(URI_1, URI_2)
	3	(X_1,sameAs,X_2) relation	explicit equality(X_1, X_2)
	4	all (parent-concepts,Y_1)	SimSet(Y_1, Y_2)
	5	(property instances,Y_1)	SimSet(Y_1, Y_2)
Property-Instances	1	(domain,X_d1) and (range,X_r1)	object equality(X_{d1}, X_{d2}), (X_{r1}, X_{r2})
	2	(parent property,Y_1)	SimSet(Y_1, Y_2)

Table 2. Features and Similarity Measures for Different Entity Types Contributing to Aggregated Similarity in NOM. The corresponding ontology is indicated through an index.

function is based on a feature (Section 4.1) of both ontologies and a respective similarity measure (Section 4.2). For NOM they are shown in Table 2. Each combination is assigned a number so we can refer to it in later sections.

4. Similarity Aggregation In our approachwe do not just aggregate the single similarity results linearly (weighted). NOM emphasizes high individual similarities and deemphasizes low individual similarities by weighting individual similarity results with a sigmoid function first and summing the modified values then. To produce an aggregated similarity (cf. Section 4.2) NOM applies

$$adj(x) = \frac{1}{1 + e^{-5(x-0.5)}}$$

Weights w_k are assigned by manually maximizing the f-measure on overall training data from different test ontologies.

5. Interpretation NOM interpretes similarity results by two means. First, it applies a threshold to discard spurious evidence of similarity. For the general threshold NOM also uses a maximized f-measure of training data. Second, NOM enforces bijectivity of the mapping by ignoring candidate mappings that would violate this constraint and by favoring candidate mappings with highest aggregated similarity scores. As there may only be one *best* mapping, every other match is a potential mistake, which is ultimately dropped.

6. Iteration The first round uses only the basic comparison method based on labels and string similarity to compute the similarity between entities. By doing the computation in several rounds one can access the already computed pairs and use more sophisticated structural similarity measures. Therefore, in the second round and thereafter NOM relies on all the similarity functions listed in Table 2.

5.2 PROMPT

PROMPT is a semi-automatic tool described in [NM03]. Together with its predecessor ONION [MWK00] it was one of the first tools for ontology merging. For this paper we concentrate on the actions performed to identify possible mapping candidates aka. merging candidates.

For this PROMPT does not require all of the steps of the process model.

1. Feature Engineering As a plug-in to Protege, PROMPT uses RDFS with features as in the previous approach.

2. Search Step Selection Like NOM, PROMPT relies on a complete comparison. Each pair of entities from ontology one and two is checked for their similarities.

3. Similarity Computation The system determines the similarities based on whether entities have similar labels. Specifically, PROMPT checks for identical labels. This is a further restriction compared to our string similarity, which also allows small deviations in the spelling.

4. Similarity Aggregation As PROMPT uses only one similarity measure, aggregation is not necessary.

5. Interpretation PROMPT presents the pairs with a similarity above a defined threshold. For these pairs chances are high that they are merged by the user. The user selects the ones he deems to be correct, which are then merged in PROMPT.

6. Iteration Iteration is done in PROMPT to allow manual refinement. After the user has acknowledged the proposition, the system recalculates the corresponding similarities and comes up with new merging suggestions.

Comparing	No.	Feature	Similarity Measure
Entity Similarity	1	(label,X_1)	explicit equality(X_1, X_2

Table 3. PROMPT: Features and Measures for Similarity

5.3 Anchor-PROMPT

Anchor-PROMPT represents an advanced version of PROMPT which includes similarity measures based on ontology structures. Only the similarity computation (step 3) changes.

3. Similarity Computation Anchor-PROMPT traverses paths between anchor points (entity pairs already identified as equal). Along these paths new mapping candidates are suggested. Specifically, paths are traversed along hierarchies as well as along other relations. This corresponds to our similarity functions based on sub- and super-concepts no. 6 and 7 and direct properties no. 4.

Comparing	No.	Feature	Similarity Measure
Concept Similarity	1	(label,X_1)	explicit equality(X_1, X_2
	4	all (direct properties,Y_1)	Set(Y_1,Y_2)
	6	all (super-concepts,Y_1)	Set(Y_1,Y_2)
	7	all (sub-concepts,Y_1)	Set(Y_1,Y_2)
Other Entity Similarity	1	(label,X_1)	explicit equality(X_1, X_2

Table 4. Anchor-PROMPT: Features and Measures for Similarity

5.4 GLUE

GLUE [DDH03] uses machine learning techniques to determine mappings.

1. Feature Engineering In a first step the Distribution Estimator uses a multi-strategy machine learning approach based on a sample mapping set. It learns a strategy to identify equal instances and concepts. Naturally a big amount of example instances is needed for this learning step.

2. Search Step Selection As in the previous approaches GLUE checks every candidate mapping.

3. Similarity Computation, 4. Similarity Aggregation, 5. Interpretation In GLUE, steps 3, 4, and 5 are very tightly interconnected, which is the reason why they are presented as one step here. The Similarity Estimator determines the similarity of two instances based on the learnt rules. From this also the mapping of concepts is derived.
Concepts and relations are further compared using Relaxation Labelling. The intuition of Relaxation Labelling is that the label of a node (in our terminology: mapping assigned to an entity) is typically influenced by the features of the node's neighborhood in the graph. The authors explicitly mention subsumption, frequency, and "nearby" nodes.A local optimal mapping for each entity is determined using the similarity results of neighboring entity pairs from a previous round. The individual constraint similarities are summarized for the final mapping probability.
Normally one would have to check all possible labelling configurations, which includes the mappings of all other entities. The developers are well aware of the problem arising in complexity, so they set up sensible partitions i.e. labelling sets with the same features are grouped and processed only once. The probabilities for the partitions are determined. One assumption is that features are independent, which the authors admit will not necessarily hold true. Through multiplication of the probabilities we finally receive the probability of a label fitting the node i.e. one entity being mapped onto another one.
From the previous step we receive the probabilities of two entities mapping onto each other. The maximum probable pair is the final mapping result.

6. Iteration To gain meaningful results only the relaxation labelling step and its interpretation have to be repeated several times. The other steps are just carried out once.

5.5 QOM — Quick Ontology Mapping

The goal of this paper is to present an efficient mapping algorithm. For this purpose, we optimize the effective, but inefficient NOM approach towards our goal. The outcome is QOM — Quick Ontology Mapping. Efficiency and complexity is described in Section 6. Here we solely present the algorithm. We would also like to point out that the efficiency gaining steps can be applied to other mapping approaches as well.

1. Feature Engineering Like NOM, QOM exploits RDF triples.

2. Search Step Selection A major ingredient of run-time complexity is the number of candidate mapping pairs which have to be compared to actually find the best mappings. Therefore, we use heuristics to lower the number of candidate mappings. Fortunately we can make use of ontological structures to classify the candidate mappings into promising and less promising pairs.

In particular we use a dynamic programming approach [Bod91]. In this approach we have two main data structures. First, we have candidate mappings which ought to be investigated. Second, an agenda orders the candidate mappings, discarding some of them entirely to gain efficiency. After the completion of the similarity analysis and their interpretation new decisions have to be taken. The system has to determine which

candidate mappings to add to the agenda for the next iteration. The behavior of initiative and ordering constitutes a search strategy.

We suggest the subsequent strategies to propose new candidate mappings for inspection:

Random A simple approach is to limit the number of candidate mappings by selecting either a fixed number or percentage from all possible mappings.

Label This restricts candidate mappings to entity pairs whose labels are near to each other in a sorted list. Every entity is compared to its "label"-neighbors.

Area Already after the first rounds some mappings have been identified. For subsequent rounds we concentrate our efforts on areas bordering to the mappings in the graph e.g. if instances have been identified to be equal, we check for their parent-concepts in the round after. In our example in the beginning of this paper (figure 1) this corresponds to: having found that *250 km/h* maps onto *fast*, we would then try to map *speed* and *characteristic*. We expect to find more mapping partners within the two close ranges.

Change Propagation QOM further compares only entities for which adjacent entities were assigned new mappings in a previous iteration. This is motivated by the fact that every time a new mapping has been found, we can expect to also find similar entities adjacent to these found mappings. Further, to prevent very large numbers of comparisons, the number of pairs is restricted.

Hierarchy We start comparisons at a high level of the concept and property taxonomy. Only the top level entities are compared in the beginning. We then subsequently descend the taxonomy. Again referring to our example: we start comparing *object* and *thing*, subsequently we continue with *vehicle* or *car* and *automobile*.

Combination The combined approach used in QOM follows different optimization strategies: it uses a label subagenda, a randomness subagenda, and a mapping change propagation subagenda. In the first iteration the label subagenda is pursued. Afterwards we focus on mapping change propagation. Finally we shift to the randomness subagenda, if the other strategies do not identify sufficiently many correct mapping candidates.

With these multiple agenda strategies we only have to check a fixed and restricted number of mapping candidates for each original entity.[6] Please note that the creation of the presented agendas does require processing resources itself.

3. Similarity Computation QOM, just like NOM, is based on a wide range of ontology feature and heuristic combinations. To keep up the high quality of mapping results we retain using as many ontology features as possible. But, in order to optimize QOM, we have restricted the range of costly features as specified in Table 5. In particular, QOM avoids the complete pair-wise comparison of trees in favor of a(n incomplete) top-down strategy. The marked comparisons in the table were changed from features which point to complete inferred sets to features only retrieving limited size direct sets.

[6] We have also explored a number of other strategies or combinations of strategies with simple data sets but they did not outperform results of QOM presented here.

Comparing	No.	Feature	Similarity Measure
Concepts	1	(label,X_1)	string similarity(X_1, X_2)
	2	(URI_1)	string equality(URI_1, URI_2)
	3	(X_1,sameAs,X_2) relation	explicit equality(X_1, X_2)
	4	(direct properties,Y_1)	SimSet(Y_1, Y_2)
	$\longrightarrow$5a	(properties of direct super-concepts, Y_1)	SimSet(Y_1, Y_2)
	$\longrightarrow$6a	(direct super-concepts, Y_1)	SimSet(Y_1, Y_2)
	$\longrightarrow$7a	(direct sub-concepts, Y_1)	SimSet(Y_1, Y_2)
	8	(concept siblings,Y_1)	SimSet(Y_1, Y_2)
	9	(direct instances,Y_1)	SimSet(Y_1, Y_2)
	$\longrightarrow$10a	(instances of direct sub-concepts, Y_1)	SimSet(Y_1, Y_2)
Relations	1	(label,X_1)	string similarity(X_1, X_2)
	2	(URI_1)	string equality(URI_1, URI_2)
	3	(X_1,sameAs,X_2) relation	explicit equality(X_1, X_2)
	4	(domain,X_d1) and (range,X_r1)	object equality(X_{d1}, X_{d2}),(X_{r1}, X_{r2})
	$\longrightarrow$5a	(direct super-properties, Y_1)	SimSet(Y_1, Y_2)
	$\longrightarrow$6a	(direct sub-properties, Y_1)	SimSet(Y_1, Y_2)
	7	(property siblings,Y_1)	SimSet(Y_1, Y_2)
	8	(property instances,Y_1)	SimSet(Y_1, Y_2)
Instances	1	(label,X_1)	string similarity(X_1, X_2)
	2	(URI_1)	string equality(URI_1, URI_2)
	3	(X_1,sameAs,X_2) relation	explicit equality(X_1, X_2)
	$\longrightarrow$4a	(direct parent-concepts, Y_1)	SimSet(Y_1, Y_2)
	5	(property instances,Y_1)	SimSet(Y_1, Y_2)
Property-Instances	1	(domain,X_d1) and (range,X_r1)	object equality(X_{d1}, X_{d2}),(X_{r1}, X_{r2})
	2	(parent property,Y_1)	SimSet(Y_1, Y_2)

Table 5. Features and Similarity Measures for Different Entity Types Contributing to Aggregated Similarity in QOM. Features with a lower case "a" have been modified for efficiency considerations.

4. Similarity Aggregation The aggregation of single methods is only performed once per candidate mapping and is therefore not critical for the overall efficiency. Therefore, QOM works like NOM in this step.

5. Interpretation Also the interpretation step of QOM is the same as in NOM. Bijectivity is enforced.

6. Iteration QOM iterates to find mappings based on lexical knowledge first and based on knowledge structures later.

In all our tests we have found that after ten rounds hardly any further changes occur in the mapping table. This is independent from the actual size of the involved ontologies. QOM therefore restricts the number of runs.

Assuming that ontologies have a fixed percentage of entities with similar lexical labels, we will easily find their correct mappings in the first iteration. These are further evenly distributed over the two ontologies, i.e. the distance to the furthest not directly found mapping is constant. Through the change propagation agenda we pass on to the next adjacent mapping candidates with every iteration step. The number of required iterations remains constant; it is independent from the size of the ontologies.

We expect that especially the restrictions imposed on steps 2. Search Step Selection and 3. Similarity Computation limit the calculational effort for mappings.

6 Comparing Run-time Complexity

We determine the worst-case run-time complexity of the algorithms to propose mappings as a function of the size of the two given ontologies. Thereby, we wanted to base our analysis on realistic ontologies and not on artifacts. We wanted to avoid the consideration of large ontologies with n leaf concepts but a depth of the concept hierarchy H_C of $n-1$. [TV03] have examined the structure of a large number of ontologies and found, that concept hierarchies on average have a branching factor of around 2 and that the concept hierarchies are neither extremely shallow nor extremely deep. The actual branching factor can be described by a power law distribution. Hence, in the following we base our results on their findings.

Proof Sketch 1 *The different algorithmic steps contributing to complexity are aligned to the canonical process of Section 3.*

*For each of the algorithms, one may then determine the costs of each step. First, one determines the cost for feature engineering (*feat*). The second step is the search step i.e. candidate mappings selection (*sele*). For each of the selected candidate mappings (*comp*) we need to compute* k *different similarity functions* sim_k *and aggregate them (*agg*). The number of entities involved and the complexity of the respective similarity measure affect the run-time performance. Subsequently the interpretation of the similarity values with respect to mapping requires a run-time complexity of* inter. *Finally we have to iterate over the previous steps multiple times (*iter*).*

Then, the worst case run-time complexity is defined for all approaches by:

$$c = (feat + sele + comp \cdot (\textstyle\sum_k sim_k + agg) + inter) \cdot iter$$

Depending on the concrete values that show up in the individual process steps the different run-time complexities are derived in detail in [ES04a].

In this paper we assume that the retrieval of a statement of an ontology entity from a database can be done in constant access time, independent of the ontology size, e.g. based on sufficient memory and a hash function.

In the following sections we will reference to this definition and mention the corresponding complexity for each action in the process.

6.1 Background Assumptions on Ontology Complexity

As background information for this section we rely on [TV03], who have examined the structure of a large number of ontologies. Ontologies typically have a branching factor b of around 2. [TV03] have also shown that the tree-like structures are neither extremely shallow nor extremely deep. All of the subsequent considerations are based on these assumptions.

We recapitulate the complexity levels which occur when doing comparisons of entity sets. n is the number of entities in the ontology. Retrieving single entities or fixed sets of entities is independent of the size of the ontology. From the constant size of all sets we say straight forward that complexity is $O(1)$. Other methods require access to a whole subtree of an ontology. The depth of an even tree leads to a complexity of $O(log(n))$. Yet even other methods need access to the whole ontology resulting in $O(n)$.

6.2 Similarity Complexity

Another issue of complexity is due to specific similarity measures. Some of the measures are considerably costly.

Object Equality requires a fixed runtime complexity $O(1)$. Comparing two individual objects (i.e. their URIs) through an equality operator is independent of the structure they are taken from.

Explicit Equality Checking the ontology whether one entity has a specific relation is also of $O(1)$.

String Equality complexity is dependent of the string size $O(length)$. But, if we assume that the length is limited by a fixed bound, complexity also is $O(1)$.

Syntactic String Similarity complexity of strings has the same considerations of fixed bounds, as for string equality ($O(1)$).

Dice For Dice one has to check for every element of the two compared sets (union) whether it is also part of the intersection. This results to a complexity of $O(|E| \cdot log(|E|) + |F| \cdot log(|F|) + (|E| + |F|))$.

SimSet has to compute the similarity vector for every entity with every other entity ($O(|E + F|^2) = O(setSize^2)$). Creating the average and normalizing does not change this complexity.

6.3 NOM - Naive Ontology Mapping

In this baseline approach we compute complex similarity methods for each possible entity pair. These are then aggregated and cleaned returning a final result.

- $feat$: No explicit feature engineering is done: $feat = 0$
- $sele$: As we check all possible mapping partners the step of choosing the mapping partners is not performed: $choice = 0$
- $comp$: All pairs of entities of the whole ontologies times themselves are processed: ($O(n^2)$).
- sim_k: The complexity of the similarity methods is determined through the number of entities which have to be explicitly accessed for a certain method. These are either single entities ($O(1)$), a fixed set of entities ($O(1)$), or in the worst case whole subtrees ($O(log(n))$). These have to be applied to the similarity measures. Table 6 shows the complexity corresponding to each rule. Critical for our case is the highest complexity i.e. $O(log^2(n))$.
- agg is done once for every entity and therefore results in $O(n)$.
- $inter$: We have to traverse the whole mapping table once more $O(n)$ for transformation of results and cleansing of duplicates (to ensure bijectivity) .
- $iter$: Several iteration rounds r are required, which itself is of $O(1)$.

We eventually receive:

$$c = (0 + 0 + O(n^2) \cdot k \cdot O(log^2(n)) + O(n) + O(n)) \cdot r = O(n^2 \cdot log^2(n))$$

6.4 PROMPT

PROMPT compares all entity pairs based on their labels. In a postprocessing step an acknowledgement of the user becomes necessary.

- $comp$: Using an ideal implementation which sorts the labels first we receive $O(n \cdot log(n))$. The tool itself requires $O(n^2)$.
- sim_k: The complexity of the similarity method is restricted to one single entity access for labels ($O(1)$).
- $inter$: The acknowledgement in the postprocessing step is done once for every entity: $inter = O(n)$
- $iteration$: PROMPT performs multiple rounds r: $O(1)$.

The total complexity is also calculated very easily.

$$c = (O(n \cdot log(n)) \cdot O(1) + O(n)) \cdot r = O(n \cdot log(n))$$

The complexity for this approach is lower than for NOM, but on the other hand only minimal features of ontologies are used for mapping computation. The consequence are results with a lower quality.

	No.	Rule	Complexity
Concepts Similarity	1	labels	single $O(1)$
	2	URIs	single $O(1)$
	3	sameAs relation	single $O(1)$
	4	direct properties	fixed set + SimSet $O(1^2)$
	5	all inherited properties	subtree+SimSet $O(log^2(n))$
	6	all super-concepts	subtree+SimSet $O(log^2(n))$
	7	all sub-concepts	subtree+SimSet $O(log^2(n))$
	8	concept siblings	fixed set + SimSet $O(1^2)$
	9	direct instances	fixed set + SimSet $O(1^2)$
	10	instances	subtree+SimSet $O(log^2(n))$
Relation Similarity	1	labels	single $O(1)$
	2	URIs	single $O(1)$
	3	sameAs relation	single $O(1)$
	4	domain and range	fixed set + SimSet $O(1^2)$
	5	all super-properties	subtree+SimSet $O(log^2(n))$
	6	all sub-properties	subtree+SimSet $O(log^2(n))$
	7	property siblings	fixed set + SimSet $O(1^2)$
	8	property instances	fixed set + SimSet $O(1^2)$
Instance Similarity	1	labels	single $O(1)$
	2	URIs	single $O(1)$
	3	sameAs relation	single $O(1)$
	4	all parent-concepts	subtree+SimSet $O(log^2(n))$
	5	property instances	fixed set + SimSet $O(1^2)$
Property-Instances Similarity	1	domain and range	fixed set + SimSet $O(1^2)$
	2	parent property	single $O(1)$

Table 6. NOM: Rules and Complexity

6.5 Anchor-PROMPT

PROMPT is enhanced by structural elements. Only the complexity of the similarity function changes.

– sim_k: This time we have to compare sub-, super-classes, and relation paths which equals subtrees: $O(log^2(n))$.

The runtime complexity changes in comparison to the original PROMPT approach to:

$$c = (O(n^2) \cdot O(log^2(n)) + O(n)) \cdot r = O(n^2 \cdot log^2(n))$$

The complexity corresponds to the complexity of the NOM approach.

6.6 GLUE

GLUE has three major steps. Preprocessing to build a classifier, the similarity determination, and the relaxation labelling.

– $feat$: The authors use machine learning techniques for the distribution estimator. An optimistic assumption could be that we use a fixed size for the training examples with a complexity of $O(1)$.
– $comp$: For every entity pair (label assignment) the probability has to be computed ($O(n^2)$).
– sim_k: If we assume a fixed size of the partitions such as being based on the number of features k, we can assume that computing the probability for one labelling configuration is done constant time $O(1)$.
– $agg, inter$: Aggregation and interpretation are part of the similarity computation and do not provide an own complexity.
– $iter$: We process several rounds, but this doesn't affect $O(1)$. Please note that only steps 3, 4, and 5 are repeated. The feature engineering aka. learning of the relaxation labeller is processed just once.

Based on the described assumptions we estimate the overall complexity as follows:

$$c = O(1) + O(n^2) \cdot O(1) \cdot r = O(n^2)$$

6.7 QOM - Quick Ontology Mapping

QOM implements different changes in comparison to the base approach NOM. We now show how this affects the outcome of complexity.

– $feat$: No transformations are required, $feat = 0$.
– $sele$: For creating the best agenda each entity is checked and a fixed set of potential comparison partners is determined. In the worst case QOM compares labels and sorts them. $sele = O(n \cdot log(n))$
– $comp$: The major efficiency leap is won through the pair reduction. As one entity only has a limited number l of other entities connected to it, only these are added to the agenda. Our complexity for this part is therefore reduced to $O(n) \cdot O(l) = O(n)$.

– sim_k: The reduced complexity of the rules is shown in table 7. Basically we have omitted the highest level of complexity - the complete subtrees. We only allow fixed sets $O(1)$.
– agg: Aggregation stays the same from NOM i.e. one action for each entity: $O(n)$.
– $inter$: Again we use the same as for NOM: transformation and cleansing once per entity: $O(n)$.
– $iter$: A constant number of rounds r is done - in our case 10. Complexity is again of $O(1)$.

	Change	Rule	Complexity
Concept Similarity	5a	properties of super-concepts	fixed set + SimSet $O(1^2)$
	6a	direct super-concepts	fixed set + SimSet $O(1^2)$
	7a	direct sub-concepts	fixed set + SimSet $O(1^2)$
	10a	instances of sub-concepts	fixed set + SimSet $O(1^2)$
Relation Similarity	5a	direct super-properties	fixed set + SimSet $O(1^2)$
	6a	direct sub-properties	fixed set + SimSet $O(1^2)$
Instance Similarity	4a	direct parent-concepts	fixed set + SimSet $O(1^2)$

Table 7. QOM: Changed Rules and Complexity

Setting the variables according to the new methods and procedures returns:

$$c = (O(n \cdot log(n)) + O(n) \cdot k \cdot O(1) + O(n) + O(n)) \cdot r = O(n \cdot log(n))$$

One can see that the theoretical complexity of this approach is much lower than in any other presented approach.

6.8 Discussion

The worst case run-time behaviors of NOM, PROMPT, Anchor-PROMPT, GLUE and QOM are given by the following table:

NOM	$O(n^2 \cdot log^2(n))$
PROMPT	$O(n \cdot log(n))$
Anchor-PROMPT	$O(n^2 \cdot log^2(n))$
GLUE	$O(n^2)$
QOM	$O(n \cdot log(n))$

We have shown that many approaches to discover mappings in ontologies pay a high calculational price for their results: NOM $O(n^2 \cdot log^2(n))$, PROMPT $O(n \cdot log(n))$, Anchor-PROMPT $O(n^2 \cdot log^2(n))$, and GLUE at least $O(n^2)$. Regardless they are very elaborated considering the quality of mapping results. But as one can easily see this

complexity is manageable neither for a real world application with large ontologies nor for small ontologies in real time environments.

We have further shown theoretically that it is possible to lower these complexities considerably. QOM only requires $O(n \cdot (log(n))$ of runtime. In the next section we show that the effectiveness of the mappings is not lower than in other approaches.

7 Empirical Evaluation and Results

In this section we show that the worst case considerations carry over to practical experiments and that the quality of QOM is only negligibly lower than the one of other approaches. The implementation itself was coded in Java using the KAON-framework[7] for ontology operations.

7.1 Test Scenario

Metrics We use standard information retrieval metrics to assess the different approaches (cf. [DMR02]):

Precision We measure the number of correct mappings found versus the total number of retrieved mappings (correct and wrong).
$p = \frac{\#correct\_found\_mapping}{\#found\_mappings}$

Recall describes the number of correct mappings found in comparison to the total number of existing mappings.
$r = \frac{\#correct\_found\_mappings}{\#existing\_mappings}$

In our tests we measure the level of f-measure reached after time. The f-measure combines the two parameters precision and recall. It was first introduced by [VR79]. The standard formula is defined as:

$$f = \frac{(b^2+1)pr}{b^2p+r}$$

b is a factor to quantify the value of precision and recall against each other. For the consequent test runs we use $b = 1$.

Data Sets Three separate data sets were used for evaluation purposes. As real world ontologies and especially their mappings are scarce, students were asked to independently create and map ontologies.[8]

Russia 1 In this first set we have two ontologies describing Russia. The students created the ontologies with the objectives to represent the content of two independent travel websites about Russia. These ontologies have approximately 400 entities each, including concepts, relations, and instances. The total number of possible mappings is 160, which the students have assigned manually. This scenario is an easy scenario, with which many individual methods can be tested.

[7] http://kaon.semanticweb.org/

[8] The datasets are available from http://www.aifb.uni-karlsruhe.de/WBS/meh/mapping/.

Russia 2 The second set again covers Russia, but the two ontologies are more difficult to map. After their creation they have been altered by deleting entities and changing the labels at random. They differ substantially in both labels and structure. Each ontology has 300 entities with 215 possible mappings, which were captured during generation. Often not even humans are capable of identifying these mappings.

Tourism Finally, the participants of a seminar created two ontologies which separately describe the tourism domain of Mecklenburg-Vorpommern. Both ontologies have an extent of about 500 entities. No instances were modelled with this ontology though, they only consist of concepts and relations. The 300 mappings were created manually.

Strategies We evaluated the mapping strategies described in the previous sections:

- PROMPT — As the PROMPT algorithm is rather simple and fast we use it as a baseline to evaluate the speed. The empirical evaluation is based on the actual implementation of PROMPT rather than its theoretic potential, as described in the previous section.
- NOM / Anchor-PROMPT — Naive Ontology Mapping is an approach making use of a wide range of features and measures. Therefore it reaches high levels of effectiveness and represents our quality baseline. In terms of structural information used and complexity incurred it is similar to Anchor-PROMPT.
- QOM — Quick Ontology Mapping is our novel approach focusing on efficiency. It uses an agenda of combined strategies as well as several other optimizing measures as described.

To circumvent the problem of having semi-automatic merging tools (PROMPT and Anchor-PROMPT) in our fully automatic mapping tests, we assumed that every proposition of the system is meaningful and correct. Further, as we had difficulties in running Anchor-PROMPT with the size of the given data sets, we refer to the results of the somewhat similar NOM. For GLUE we face another general problem. The algorithm has a strong focus on example instance mappings. As we can not provide this, we refrained from running the tests on a poorly trained estimator which would immediately result in poor quality results.

7.2 Results and Discussion

We present the results of the strategies on each of the data sets in Figures 3 to 5. The x-axis shows the elapsed time on a logarithmic scale, the y-axis corresponds to the f-measure. The symbols represent the result after each iteration step.

Depending on the scenario PROMPT reaches good results within a short period of time. Please notice that for ontologies with a small number of similar labels (Figure 4) this strategy is not satisfactory (f-measure 0.06). In contrast, the f-measure value of the NOM strategy rises slowly but reaches high absolute values of up to 0.8. Unfortunately it requires a lot of time. Finally the QOM Strategy is plotted. It reaches high quality levels very quickly. In terms of absolute values it also seems to reach the best quality results of all strategies. This appears to be an effect of QOM achieving an about 20 times higher number of iterations than NOM within the given time frame.

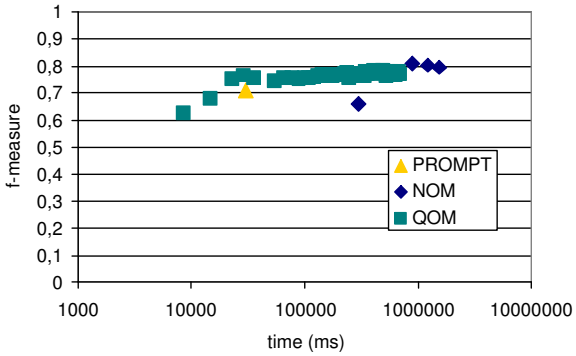

Fig. 3. Mapping quality reached over time with Russia 1 ontologies.

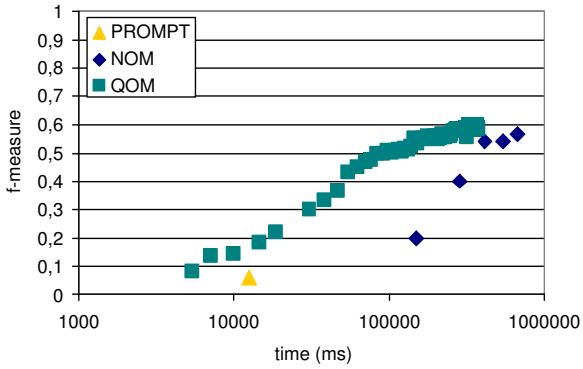

Fig. 4. Mapping quality reached over time with Russia 2 ontologies.

Lessons Learned. We had the hypothesis that faster mapping results can be obtained with only a negligible loss of quality. We here briefly present the bottom line of our considerations in this paper:

1. Optimizing the mapping approach for efficiency — like QOM does — decreases the overall mapping quality. If ontologies are not too large one might prefer to rather avoid this.
2. Labels are very important for mapping, if not the most important feature of all, and alone already return very satisfying results.
3. Using an approach combining many features to determine mappings clearly leads to significantly higher quality mappings.
4. The Quick Ontology Mapping approach shows very good results. Quality is lowered only marginally, thus supporting our hypothesis.

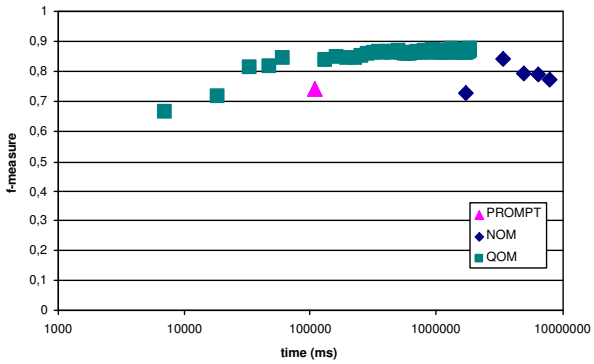

Fig. 5. Mapping quality reached over time with Tourism ontologies.

5. QOM is faster than standard prominent approaches by a factor of 10 to 100 times.

Recapitulating we can say that our mapping approach is very effective and efficient.

8 Related Work

Most of the ideas for measuring similarity are derived from common sense and are easily understood. To our knowledge existing approaches focus on specific methods to determine similarity rather than using an overall integrating and efficient approach.

Various authors have tried to find a general description of similarity with several of them being based on knowledge networks. [RE00] give a general overview of similarity.

As the basic ontology mapping problem has been around for some years, first tools have already been developed to address this. The tools PROMPT and AnchorPROMPT [NM03] use labels and to a certain extent the structure of ontologies. However, their focus lies on ontology merging i.e. how to create one ontology out of two. Potential matches are presented to the user for confirmation. In their tool ONION [MWK00] the authors use rules and inferencing to execute mappings, but is based on manually assigned mappings or very simple heuristics. [DDH03] use a general approach of relaxation labelling in their tool GLUE. However, most of their work is based on the similarity of instances only. [McG00] created a tool for mapping called Chimaera. Besides equality first steps are taken in the direction of complex matches. These could also include concatenation of two fields such as “first name” and “last name” to “name”[DR02]. Another interesting approach for schema and ontology mapping is presented by [BMSZ03]. Explicit semantic rules are added for consideration. A SAT solver is used to prevent mappings to imply semantical contradictions.

Despite the large number of related work on effective mapping already mentioned throughout this paper, there are very few approaches raising the issue of efficiency.

Apart from the ontology domain research on mapping and integration has been done in various computer science fields. [AS01] present an approach to integrate

documents from different sources into a master catalog. There has also been research on efficient schema and instance integration within the database community. [RHdV03,MNU00,YMK02] are a good source for an overview. However, even though these algorithms have been optimized for many years, they can only be partly used for our purposes, as they are mainly oriented towards (domain-specific) instance comparisons rather than schema matching. Due to this comparisons with our approach are very difficult. Another community involved in similarity and mapping are object-oriented representations[BS98]. To the best of our knowledge, the OO community has explored efficient similarity computations to very little extent. [Ruf03] shows an approach for UML. Even though efficiency has been a topic in related areas, only very little can directly be transferred to ontology mapping.

9 Conclusion

The problem of mapping two ontologies effectively and efficiently arises in many application scenarios [EHvH$^+$03,HSS03]. We have devised a generic process model to investigate and compare different approaches that generate ontology mappings. In particular, we have developed an original method, QOM, for identifying mappings between two ontologies. We have shown that it is on a par with other good state-of-the-art algorithms concerning the quality of proposed mappings, while outperforming them with respect to efficiency — in terms of run-time complexity ($O(n \cdot log(n))$ instead of $O(n^2)$) and in terms of the experiments we have performed (by a factor of 10 to 100). Vice versa QOM shows better quality results than approaches within the same complexity class.

Acknowledgements Research reported in this paper has been partially financed by the EU in the IST projects WonderWeb (IST-2001-33052), SWAP (IST-2001-34103) and SEKT (IST-2003-506826).

References

[AS01] Rakesh Agrawal and Ramakrishnan Srikant. On integrating catalogs. In *Proceedings of the tenth international conference on World Wide Web*, pages 603–612. ACM Press, 2001.

[Bis95] G. Bisson. Why and how to define a similarity measure for object based representation systems. *Towards Very Large Knowledge Bases*, pages 236–246, 1995.

[BMSZ03] Paolo Bouquet, B. Magnini, L. Serafini, and S. Zanobini. A SAT-based algorithm for context matching. In *IV International and Interdisciplinary Conference on Modeling and Using Context (CONTEXT'2003)*, Stanford University (CA, USA), June 2003.

[Bod91] M. Boddy. Anytime problem solving using dynamic programming. In *Proceedings of the Ninth National Conference on Artificial Intelligence*, pages 738–743, Anaheim, California, 1991. Shaker Verlag.

[BS98] Ralph Bergmann and Armin Stahl. Similarity measures for object-oriented case representations. *Lecture Notes in Computer Science*, 1488:25+, 1998.

[CAFP98] S. V. Castano, M. G. De Antonellis, B. Fugini, and C. Pernici. Schema analysis: Techniques and applications. *ACM Trans. Systems*, 23(3):286–333, 1998.

[CC94] T. Cox and M. Cox. *Multidimensional Scaling*. Chapman and Hall, 1994.

[DDH03] A. Doan, P. Domingos, and A. Halevy. Learning to match the schemas of data sources: A multistrategy approach. *VLDB Journal*, 50:279–301, 2003.

[DLD+04] Robin Dhamankar, Yoonkyong Lee, AnHai Doan, Alon Halevy, and Pedro Domingos. imap: discovering complex semantic matches between database schemas. In *Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, pages 383–394, 2004.

[DMR02] H. Do, S. Melnik, and E. Rahm. Comparison of schema matching evaluations. In *Proceedings of the second int. workshop on Web Databases (German Informatics Society)*, 2002.

[DR02] H. Do and E. Rahm. COMA - a system for flexible combination of schema matching approaches. In *Proceedings of the 28th VLDB Conference*, Hong Kong, China, 2002.

[EHvH+03] M. Ehrig, P. Haase, F. van Harmelen, R. Siebes, S. Staab, H. Stuckenschmidt, R. Studer, and C. Tempich. The SWAP data and metadata model for semantics-based peer-to-peer systems. In *Proceedings of MATES-2003. First German Conference on Multiagent Technologies*, LNAI, Erfurt, Germany, September 22-25 2003. Springer.

[ES04a] M. Ehrig and S. Staab. Quick ontology mapping with QOM. Technical report, University of Karlsruhe, Institute AIFB, 2004. http://www.aifb.uni-karlsruhe.de/WBS/meh/mapping/.

[ES04b] Marc Ehrig and York Sure. Ontology mapping - an integrated approach. In Christoph Bussler, John Davis, Dieter Fensel, and Rudi Studer, editors, *Proceedings of the 1st ESWS*, volume 3053 of *Lecture Notes in Computer Science*, pages 76–91, Heraklion, Greece, MAY 2004. Springer Verlag.

[EV03] J. Euzenat and P. Valtchev. An integrative proximity measure for ontology alignment. In Anhai Doan, Alon Halevy, and Natasha Noy, editors, *Proceedings of the Semantic Integration Workshop at ISWC-03*, 2003.

[Gru93] Tom R. Gruber. Towards Principles for the Design of Ontologies Used for Knowledge Sharing. In N. Guarino and R. Poli, editors, *Formal Ontology in Conceptual Analysis and Knowledge Representation*, Deventer, The Netherlands, 1993. Kluwer Academic Publishers.

[HSS03] A. Hotho, S. Staab, and G. Stumme. Ontologies improve text document clustering. In *Proceedings of the International Conference on Data Mining — ICDM-2003*. IEEE Press, 2003.

[Lev66] I. V. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. *Cybernetics and Control Theory*, 1966.

[McG00] Deborah L. McGuinness. Conceptual modeling for distributed ontology environments. In *International Conference on Conceptual Structures*, pages 100–112, 2000.

[MMSV02] Alexander Maedche, Boris Motik, Nuno Silva, and Raphael Volz. Mafra - a mapping framework for distributed ontologies. In *Proceedings of the EKAW 2002*, 2002.

[MNU00] Andrew McCallum, Kamal Nigam, and Lyle H. Ungar. Efficient clustering of high-dimensional data sets with application to reference matching. In *Knowledge Discovery and Data Mining*, pages 169–178, 2000.

[MS02] Alexander Maedche and Steffen Staab. Measuring similarity between ontologies. In *Proceedings of the European Conference on Knowledge Acquisition and Management (EKAW)*. Springer, 2002.

[MW01] Prasenjit Mitra and Gio Wiederhold. An ontology-composition algebra. Technical report, Stanford University, Stanford, California, USA, 2001.

[MWK00] Prasenjit Mitra, Gio Wiederhold, and Martin Kersten. A graph-oriented model for articulation of ontology interdependencies. *Lecture Notes in Computer Science*, 1777:86+, 2000.

[NM03] Natalya F. Noy and Mark A. Musen. The PROMPT suite: interactive tools for ontology merging and mapping. *International Journal of Human-Computer Studies*, 59(6):983–1024, 2003.

[RE00] M. Andrea Rodrguez and Max J. Egenhofer. Determining semantic similarity among entity classes from different ontologies. *IEEE Transactions on Knowledge and Data Engineering*, 2000.

[RHdV03] John Roddick, Kathleen Hornsby, and Denise de Vries. A unifying semantic distance model for determining the similarity of attribute values. In *Proceedings of the 26th Australsian Computer Science Conference (ACSC2003)*, Adelaide, Australia, 2003.

[Ruf03] Raimi Ayinde Rufai. Similarity metric for uml models. Master's thesis, King Fahd University of Petroleum and Minerals, 2003.

[SEH+03] Gerd Stumme, Marc Ehrig, Siegrfried Handschuh, Andreas Hotho, Alecander Maedche, Boris Motik, Daniel Oberle, Christoph Schmitz, Steffen staab, Ljiljana Stojanovic, Nenad Stojanovic, Rudi Studer, York Sure, Raphael Volz, and Valentin Zacharias. The Karlsruhe view on ontologies. Technical report, University of Karlsruhe, Institute AIFB, 2003.

[She00] Colin Shearer. The CRISP-DM model: The new blueprint for data mining. *Journal of Data Warehousing*, 2000.

[Stu02] H. Stuckenschmidt. Approximate information filtering with multiple classification hierarchies. *International Journal on Computational Intelligence Applications*, 2002. Accepted for publication.

[Su02] Xiaomeng Su. A text categorization perspective for ontology mapping. Technical report, Department of Computer and Information Science, Norwegian University of Science and Technology, Norway, 2002.

[TV03] Christoph Tempich and Raphael Volz. Towards a benchmark for semantic web reasoners - an analysis of the DAML ontology library. In York Sure, editor, *Evaluation of Ontology-based Tools (EON2003) at Second International Semantic Web Conference (ISWC 2003)*, October 2003.

[VR79] C. J. Van Rijsbergen. *Information Retrieval, 2nd edition*. Dept. of Computer Science, University of Glasgow, 1979.

[YMK02] Liu Yaolin, Martin Molenaar, and Menno-Jan Kraak. Semantic similarity evaluation model in categorical database generalization. In *Symposium on Geospatial Theory*, 2002.