An Iterative Algorithm for Ontology Mapping Capable of Using Training Data

Andreas Heß

Vrije Universiteit Amsterdam University College Dublin
andreas@few.vu.nl

Abstract. We present a new iterative algorithm for ontology mapping where we combine standard string distance metrics with a structural similarity measure that is based on a vector representation. After all pairwise similarities between concepts have been calculated we apply well-known graph algorithms to obtain an optimal matching. Our algorithm is also capable of using existing mappings to a third ontology as training data to improve accuracy. We compare the performance of our algorithm with the performance of other alignment algorithms and show that our algorithm can compete well against the current state-of-the-art.

1 Introduction

In this paper, we present an iterative algorithm[1] for ontology mapping that is based on established string distance metrics that have been discussed in literature and on a structural similarity measure that is based on a vector representation of the relations between entities. Furthermore, we show how we can use a given mapping to a third ontology as training data or background knowledge to improve mapping accuracy.

The remainder of this paper is structured as follows: We start with a formal problem formulation in section 2. We continue with a short discussion of related work (section 3). Then, we present our algorithm in detail in section 4. We discuss various parameters and design choices in section 5. Finally, we evaluate the performance of different configurations of our algorithm and show that we can compete well against the current state-of-the-art (see section 6) before we conclude in section 7.

2 Problem Formulation

In various approaches such as [12], the schema mapping problem is cast as a graph matching problem. We follow that notion and treat the entities that we

[1] An implementation of our algorithm called *dam*^{2} is available at `http://www.few.vu.nl/~andreas/projects/dam2/`

are trying to match as nodes in a graph.[2] Therefore, we use the words "node", "vertex" and "entity" synonymously.

We define the mapping problem as identifying pairs of vertices from two edge-labelled directed graphs. Vertices represent entities in the ontology (i.e. classes and properties). The arcs denote relations between these entities, and the labels signify the kind of relation, e.g. "subclass of", "domain" or "range".

Definition 1. *Let $G = (V, A)$ and $G' = (V', A')$ be two directed graphs with V and V' as their set of vertices and A and A' as their set of arcs. We define two partial functions as:*

$$\text{map} : V \rightharpoonup V' \qquad \text{map}' : V' \rightharpoonup V$$

The ontology alignment problem formally consists of finding a number of mappings $v' = \text{map}(v)$ for as many $v \in V, v' \in V'$ as possible. We restrict ourselves to finding mappings for classes and properties only. We define the sets of classes and properties as subsets of V resp. V' and only map classes to classes and properties to properties. Furthermore, we restrict the mapping function to being injective (but not necessarily surjective), i.e. we restrict ourselves to one-to-one-mappings.

We split the problem into two parts: First, we define a similarity function:

Definition 2. *To measure the similarity between vertices from G and G', we define two similarity functions as:*

$$\text{sim} : V \times V' \to [0, 1] \qquad \text{sim}' : V' \times V \to [0, 1]$$

We make use of different similarity functions that we denote with indices. We use this similarity function to compute all pairwise similarities between all $v \in V$ and $v' \in V'$. Section 4 describes this step in detail.

The second part of the problem is to convert these pairwise similarities into mappings. We treat the pairwise similarities as a bipartite graph $B = (V + V', E)$ with the entities from V and V' as nodes and a weighted edge where the similarity between two entities $\text{sim}(v, v') > 0$. The problem of obtaining the *map*-relation is then equivalent to the problem of finding a matching in this bipartite graph. We do in general not require that $\text{sim}(v, v') = \text{sim}'(v', v)$. In that case, the edges in the bipartite graph B are directed and the weights are not symmetric. Section 4.5 describes the application of two well-known graph-algorithms to this problem.

3 Related Work

While many approaches have been proposed for schema matching in the past (e.g. Cupid [11]), dedicated algorithms for ontology matching are newer. Among

[2] The paper mentioned, [12], but also newer algorithms such as [9] make use of derived graphs or alternative representations such as the pairwise connectivity graph and the similarity propagation graph in [12] or a bipartite graph representation of RDF as in [9]. We do not use such derived graphs in our approach.

these are for example QOM [2] (which is optimised for speed) and OLA [4], which combines a variety of different similarity measures. A very recent development is the Falcon algorithm [9] that has been shown to perform very well.

The level of competition that came along with these different approaches has led to ontology alignment contests. Such contests have taken place at the Information Interpretation and Integration Conference (I^3CON) in 2003, the Third International Workshop on Evaluation of Ontology Based Tools in 2004 [15] and at the Third International Conference on Knowledge Capture (K-CAP 2005) [5]. In section 6, we will compare our own algorithm to those presented at the latter event.

Following [14], we refer to similarity measures that are based on inherent characteristics of an entity as *intrinsic*, where as all structural similarity measures that are based on relations to other entities are referred to as *extrinsic*. Most mapping algorithms adhere to a simple structure: an initial calculation of an intrinsic similarity measure is followed by an iterative calculation of an extrinsic measure before finally the mappings are derived from the pairwise similarities. Ehrig and Staab discuss this structure in greater detail in [2]. Our algorithm adheres to this common structure, too. However, there are two features which make it distinct from other algorithms that we are aware of. The first point where our algorithm differs from others is the way how the extrinsic similarity is computed. In a variety of approaches, extrinsic similarity is basically just the propagated intrinsic similarity of the neighbouring entities. In our approach, we compute extrinsic similarity by using a feature vector. Section 4.2 describes the details.

The second important feature is the way how the similarities are transformed into mappings. While Melnik et al. in [12] propose to compute either a stable marriage or the maximum weighted matching in a bipartite graph to find a good mapping, it seems that most newer ontology mapping algorithms do not do this (e.g. Ehrig and Staab use a simple greedy approach in [2]). In section 4.5 we describe how these two well-known graph algorithms can be used, and an empirical evaluation (see section 6) shows that it increases the performance of the mapping algorithm.

4 The Algorithm

4.1 Computing Intrinsic Similarity

In our implementation, we use distance metrics from the well-known SecondString library[3] as intrinsic similarity measures.

We also experimented with a similarity measure based on WordNet.[4] We used a similarity metric based on Euzenat's implementation in the OWL alignment API [3]. We decided, however, not to use it in the current setup. Preliminary experiments suggested that on many datasets no or only a marginal improvement

[3] `http://secondstring.sourceforge.net/`, see also [1]

[4] `http://wordnet.princeton.edu/`

can be achieved. This small benefit is, however, contrasted by a much greater computational effort. It may be possible to overcome these limitations by using a more sophisticated algorithm for computing a semantic similarity based on WordNet. This is, however, deferred to future work.

We use URIs, labels, comments and text from individuals and property values as text sources. We conducted experiments with the Jaro-Winkler metric [16] and a version of Levenshtein edit distance [10] that is scaled to the range $[0, 1]$ for comparing labels and local names. We used a soft-token metric with Jaro-Winkler resp. scaled Levenshtein edit distance as the base string distance metric for comparing comments and instance data. To determine the overall intrinsic similarity between two concepts, we use the maximum of these metrics. To avoid overemphasising small similarities, we disregard similarities that are smaller than a threshold of 0.4 and map similarities greater than 0.4 to the full range $[0, 1]$.

4.2 Computing Extrinsic Similarity

The main difference between our approach and existing schema matching algorithms is the way how the extrinsic similarity is computed. In many previous approaches extrinsic or structural similarity is propagated through a graph structure that is determined by the schema or ontology.

In our approach, we use an extrinsic feature vector $\mathbf{de}(v)$ for each entity that captures the relationship between this and other entities and then compute the similarities between these vector representations. The intuition behind using this vector representation is analogous to the assumption that the propagation-based methods make: Two nodes are similar if they are related to similar nodes.

To formally define the extrinsic feature vector, we first have to introduce a function that computes all entities that are connected to an entity v by a relation l. We consider for example subsumption and domain and range of properties as relations.

Definition 3. *We define a function from the set of vertices and the set of labels L to the power set of vertices so that for a given vertex the function finds all vertices adjacent through an arc with a given label:*

$$\mathrm{rel} : V \times L \to 2^V$$

Let $G = (V, A)$ be a digraph with the set of vertices V and labelled arcs A as a set of ordered triples $(v, w, l) \in V \times V \times L$. Then we define:

$$\mathrm{rel}(v, l) = \{x | v, x \in V \wedge (v, x, l) \in A\}$$

The definition of $\mathrm{rel}' : V' \times L \to 2^{V'}$ is analogous.

Next, as an intermediate step to our extrinsic feature vector function, we define a *dynamic intrinsic* feature vector function that is basically a vector representation of all similarities between an entity v and all entities $v' \in V'$. "Intrinsic" means that these features are inherent to an entity. "Dynamic" means

that their value can change as we get more information about that entity, and can thus make a better prediction about the similarities between this and other entities. Note that the dynamic intrinsic features are typically what we want to compute. In particular, this means that the dynamic intrinsic features are initially unknown.

Definition 4. *We define a dynamic intrinsic feature vector function as a function of an entity:*

$$\mathbf{di} : V \to \mathbb{R}^{|V'|}$$

Analogous to the matrix representation of a graph, we impose an arbitrary total order on V' and denote the first element of V' as v'_0 and the subsequent elements as v'_n for all $n < |V'|$. Then we define **di** *as follows:*

$$\mathbf{di}(v) = [\mathrm{sim}(v, v'_0), \mathrm{sim}(v, v'_1), \ldots, \mathrm{sim}(v, v'_{|V'|-1})]$$

Dynamic extrinsic features are dynamic intrinsic features of related entities:

Definition 5. *We define a dynamic extrinsic feature vector function as a function of an entity.*

$$\mathbf{de} : V \to \mathbb{R}^{|V'|}$$

Assuming a commutative and associative operator $\oplus$ on $\mathbb{R}^d$ and a function rel *as per definition 3, we define* $\mathbf{de}(v)$ *as some combination $\oplus$ of the dynamic intrinsic features* $\mathbf{di}(x)$ *(see definition 4) of all related entities $x \in \mathrm{rel}(v)$.*

$$\mathbf{de}(v) = \bigoplus_{x \in \mathrm{rel}(v)} \mathbf{di}(x)$$

The exact definition of the combination operator $\oplus$ is arbitrary. We use an additive operator in our experiments.

Note that the elements in $\mathbf{de}(v)$ are based on the relations of $v \in V$, but correspond to vertices in V'. In order to compute an extrinsic similarity between v and some v', we have to define an extrinsic feature vector for v' that is based on the relations of $v' \in V'$.

Definition 6. *We define an extrinsic feature vector function as a function of an entity:*

$$\mathbf{de'} : V' \to \mathbb{R}^{|V'|}$$

Based on the total order on V' from definition 4, we define that each element i in $\mathbf{de'}$ *is* 1*, if $v'_i \in \mathrm{rel}(v')$ and* 0 *otherwise.*

Given definitions 5 and 6 we can now easily define an extrinsic similarity $\mathrm{sim}_{\mathrm{ext}}(v, v')$ based on the similarity between the vectors $\mathbf{de}(v)$ and $\mathbf{de'}(v')$. A common similarity measure for two vectors is the dot product, but it is usually better to normalise the similarity measure using the well-known cosine, Dice, Jaccard or overlap coefficients, which are widely used in information retrieval. The similarities based on the extrinsic feature vectors are not symmetric. Since the feature vector is based on the best mapping for each concept, the fact that v maps to v' does not necessarily mean that the best mapping for v' is v, if the overall similarity $\mathrm{sim}(v, v')$ is greater than the similarity of v to all other $x' \in V'$ but less than the similarity $\mathrm{sim}(v', x)$ of v' to some $x \in V$.

Algorithm 1 Iterative Similarity Calculation

for $v \in V$ **do**
  $\mathbf{di}_{\text{int}}(v) \leftarrow [\text{sim}_{\text{int}}(v, v'_0), \text{sim}_{\text{int}}(v, v'_1), \ldots, \text{sim}_{\text{int}}(v, v'_{|V'|-1})]$
end for
/ Initially, use intrinsic similarity only */*
$\mathbf{de}(v) \leftarrow \bigoplus_{x \in \text{rel}(v)} \mathbf{di}_{\text{int}}(x)$
for a fixed number of iterations **do**
  for $v \in V$ **do**
    $\mathbf{di}_{\text{ext}}(v) \leftarrow [\text{sim}_{\text{ext}}(v, v'_0), \text{sim}_{\text{ext}}(v, v'_1), \ldots, \text{sim}_{\text{ext}}(v, v'_{|V'|-1})]$
    / Combine intrinsic and extrinsic similarity */*
    $\mathbf{di}(v) \leftarrow \mathbf{di}_{\text{int}}(v) \otimes \mathbf{di}_{\text{ext}}(v)$
  end for
  $\mathbf{de}(v) \leftarrow \bigoplus_{x \in \text{rel}(v)} \mathbf{di}(x)$
end for
return $\forall v \in V : \mathbf{di}(v)$

4.3 Iterative Algorithm

Algorithm 1 formally specifies the iterative method of calculating the overall similarity. We are not restricted to computing $\text{sim}(v, v')$, calculating $\text{sim}(v', v)$ is analogous. Recall that because of the way we the extrinsic similarity is defined they are not necessarily equal.

4.4 Using Training Data

It is quite straightforward to use a previously known mapping to a third ontology to improve mapping accuracy. We assume a third ontology V'' and known mappings of the form (v', v'') with $v' \in V'$ and $v'' \in V''$. We compute the pairwise similarities between all $v \in V$ and both $v'' \in V''$ and $v' \in V'$ as in algorithm 1. Then, for each pair (v', v'') we assume the maximum[5] of $\text{sim}(v, v')$ and $\text{sim}(v, v'')$ as the joint similarity and substitute the similarity values in the mapping algorithm with the joint similarity. Let v'' be the entity that is mapped to v' as by the background knowledge. Then, we substitute the assignments of **di** in algorithm 1 with:

$$\mathbf{di}(v) \leftarrow [\max(\text{sim}(v, v'_0), \text{sim}(v, v''_0)), \ldots, \max(\text{sim}(v, v'_{|V'|-1}), \text{sim}(v, v''_{|V'|-1}))]$$

Note that this replacement takes places for both the intrinsic and the extrinsic similarity and therefore the subscript has been omitted.

It is of course in principle also possible to use more than one known mapping as training data, but for our experiments we restricted ourselves to cases with a known mapping to just one other ontology. Furthermore, it would be possible to replace the entire similarity function with the output of a machine learning algorithm. In fact, algorithm 1 is very similar to the supervised learning algorithm that we presented in [8] and could be seen as a generalisation thereof.

[5] Other ways of defining the joint similarity, for example using the average, are thinkable. Using the maximum is like using a nearest-neighbour classifier.

4.5 Postprocessing Steps

Once we have computed the overall similarities, we have to compute the actual one-to-one mapping. This is the problem of finding a matching in a bipartite graph. A bipartite graph $B = (V + V', E)$ is a graph where the nodes can be split in two groups such that every edge connects two nodes from both partitions. Every similarity that has been calculated in the previous step corresponds to a weighted edge in such a bipartite graph.[6] A matching M in a graph is a set of edges such that no node is incident to more than one edge. In our setting this corresponds to a one-to-one mapping: For every entity in one ontology we want to find one entity in the other ontology. M is called maximum-weighted, if there is no other matching where the sum of all edge weights in the matching is bigger. M is called a stable marriage, if there are no nodes $v \in V$ and $v' \in V'$ such that the edge between v and v' in B is not in M, but has a higher weight than the edges in M that are incident in v and v'.

Gale and Shapley have shown in [6] that for bipartite graphs where the two partitions are of the same size a stable marriage always exists and presented an algorithm for finding such a matching. Because the number of vertices in V and V' is not necessarily equal in our case (if the two ontologies are of different size), a perfect match (in the graph-theoretic sense) is not always possible. It is therefore necessary to modify the termination criterion of the original Gale/Shapley algorithm slightly in a way that is equivalent to adding pseudo nodes and edges with weight zero to the bipartite graph.

Melnik et al. in [12] propose to compute either a stable marriage or the maximum weighted matching to find a good mapping. We compared the two approaches empirically on our data. We used an off-the-shelf implementation of James Munkres' algorithm [13] (also referred to as the Hungarian algorithm) to compute maximum-weighted matchings. As opposed to the Gale/Shapley algorithm, Munkres' algorithm is not suited for graphs with directed edges and asymmetric weights. Due to the way the extrinsic similarity is computed, the edge weights are not necessarily symmetric in our case, but it is of course straightforward to create a graph with undirected edges and symmetric weights simply by addition.

5 Parameters

Our matching algorithm as presented in this paper has various parameters.

In this section, we discuss various options and design choices. The experiments in section 6 show empirically the influence of the parameter settings on the overall performance.

[6] Note that this bipartite graph must not be confused with the graph interpretation of the two ontologies! We use a bipartite graph only to determine the final matching once the pairwise similarities have been calculated.

5.1 Structure

Depending on the expressiveness of the underlying ontology language, several relations between classes or properties are defined. We considered super- and subclass-relations, super- and subproperties, defined properties for a class, domain and range of a property and siblings of classes and properties as possible relations. We performed preliminary experiments on three different combinations of those features: First, we used all available relations. Second, we used all relations except for siblings, and third, we used the subsumption relation only. These experiments have shown that the second setup consistently performed best. In our final evaluation (see next section), we call this configuration "dublin2"[7].

5.2 Number of Iterations

As in our experiments with iterative ensemble classification, we decided to use a fixed number of iterations as termination criterion for reasons of simplicity, and because it is not proven that the algorithm converges. Preliminary empirical experiments suggested that the algorithm is not very sensitive to the exact number of iterations. We set the number of iterations to five, the same as in our earlier work on web service annotation [8].

5.3 Inference

When mapping rich ontologies, it is sometimes possible to exploit knowledge drawn from the ontologies itself to impose constraints on the mappings or to infer mappings. Although we believe that for some mapping tasks exploiting such knowledge could increase the mapping accuracy, such an approach is out of scope of this thesis. We restrict ourselves to using the information obtained through the iterative relational algorithm to compute the final mappings. The set of ontologies we used for evaluating our algorithm does not have a very rich structure, so in comparison with other algorithms that may use such inference, our algorithm has no disadvantage.

5.4 Post-processing

As discussed above, we have to consider at least two ways of creating a mapping from the acquired similarities, if we demand a one-to-one mapping. We can compute either a stable marriage or a maximum weighted matching. In our empirical experiments, we tried both approaches. In the graphs and tables presenting our results we denote configurations that use the Gale/Shapley algorithm (as opposed to a maximum weighted matching) with the letter "g".

We also tried both possible answers to the question when the post-processing step should be applied. We denote the configurations where we applied the post-processing step also in between iterations with the letter "e". In the other experiments, the post-processing step (i.e. applying the Gale/Shapley or Hungarian

[7] The OAEI 2005 contest lists our results as "dublin", we keep the name here.

algorithm) was only performed after the iteration phase of the algorithm has been completed.

5.5 Intrinsic Similarity

We already discussed the way we compute the intrinsic similarity between two concepts above in section 4.1. However, we could plug an arbitrary string distance metric in our framework. A great variety of string distance metrics – established algorithms as well as ad-hoc measures – is available off-the-shelf in libraries such as the already mentioned SecondString. As mentioned above, we considered the Jaro-Winkler and Levenshtein metrics. Preliminary experiments have shown that with our data, a scaled version of the Levenshtein metric works generally better than Jaro-Winkler. Therefore, we decided to use only the scaled Levenshtein metric in our final experiments. We set the threshold for the soft-token metric to 0.9, i.e. two tokens that have a string similarity greater or equal than 0.9 are considered the same. The suitability of different string distance metrics for several tasks has been extensively discussed in literature, e.g. [1].

5.6 Thresholds

In order to avoid spurious mappings it makes sense to use a minimum similarity threshold. In the ontology mapping scenario, it is not guaranteed that for some concept in one ontology a concept in another ontology actually exists. In these cases, not making a prediction is the correct answer. But also in other cases it is in several scenarios useful not to make a prediction at all rather than making a bad prediction. For example, consider a semi-automated setting where a human annotator has to review suggestions made by the algorithm.

For the precision/recall-graphs, we varied the threshold between 0 and 1 in steps of 0.05. When comparing the different configurations of our algorithm and for comparison with the algorithms from the OAEI 2005 contest we used a zero threshold.

6 Evaluation

We evaluated our algorithm on the benchmark ontologies from the 2005 Ontology Alignment Evaluation Initiative (OAEI 2005, [5]). Most of the benchmark ontologies consist of versions of a base ontology, where different aspects have been changed. For most of the following tests, we concentrate on six interesting ontologies: In two cases (ontologies 205 and 206 from the test suite), all names and labels have been replaced with synonyms or foreign words, and in four cases, independently developed "real-world" ontologies that describe the same domain have been used (301-304).

We tested various configurations of our algorithm and compared the results from these different setups against each other as well as against the published results from the other participants of the contest. The experiments were conducted in order to answer the five basic (groups of) questions:

1. Do we get any benefit from the extrinsic features as opposed to using the intrinsic similarity only?
2. Is it better to compute the maximum weighted matching or is a stable marriage more important? Should we apply this step only after all similarities are computed, or also between iterations?
3. What threshold is optimal?
4. How does our algorithm perform compared to other algorithms in literature? What are the strengths and weaknesses?
5. What is the benefit of using known mappings as training data?

It is important to note that in most of the experiments the difference in performance between the different configurations was quite low, although there are visible trends. However, what the experiments clearly show is that the overall accuracy of ontology mapping is based largely on the initial intrinsic (lexical) mapping. Unfortunately, because it is rarely published what the contributions of the lexical and structural similarities are, it is difficult to compare the strengths and weaknesses to other algorithms. Space restrictions prevent us to present all our results here. For a more detailed discussion, the reader is referred to [7].

6.1 Extrinsic vs. Intrinsic Features

The first question is of course the most crucial one: Is the way how we use the additional relational information, that differs from other methods known in literature, useful? Does it work? To answer this question, we compared the "dublin10" setup with the "dublin2e0" and "dublin20" setup. The "dublin10" setup uses only intrinsic features, "dublin20" and "dublin2e0" use extrinsic features (in five iterations) as well. Both setups compute a maximum-weighted matching, the "dublin2e" configuration also enforces one-to-one mappings in between iterations.

The results in figure 1 (note that the scale starts with 0.4 to emphasise the difference between configurations) show that on four ontologies the configuration that uses extrinsic features performs better or equal than the configuration with only the intrinsic features. However, in two of the "real-world" ontologies, using the extrinsic features affects the performance in a negative way. The reason for this is that the ontologies 303 and 304 are structurally different from the base ontology and our algorithm is mislead by this structural difference. In that case, any attempt to make predictions based on the structure must fail. The other four ontologies, especially 205 and 206, are structurally quite similar to the base ontology. Here using the extrinsic features helps.

We conclude from these results that using relational features can improve the performance, but only if the ontologies that are to be matched are not structurally different. This is not only true for our approach. For example, in [9], the authors make the same observation for the Falcon algorithm.

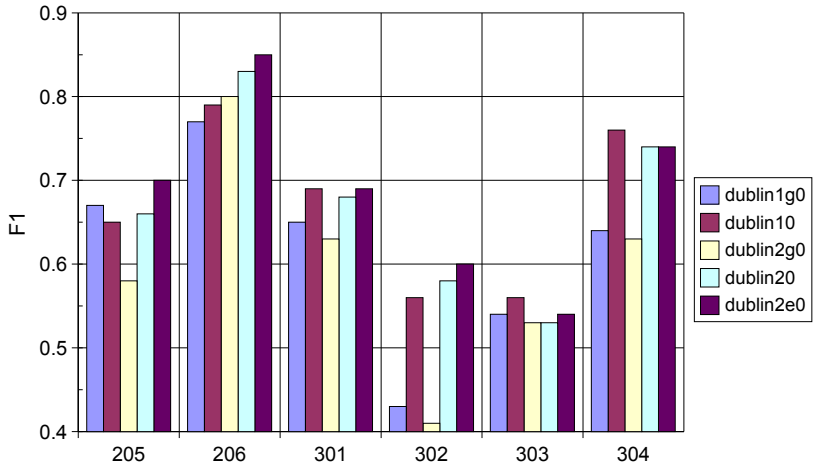

Fig. 1. Comparison of different configurations of our algorithm

6.2 Stable Marriage vs. Maximum-Weighted Matching

As far as we are aware, most other current algorithms do not explicitly compute stable marriages or maximum-weighted matchings to determine a one-to-one mapping. The Similarity Flooding algorithm [12] is a notable exception. We compared configurations that use a stable marriage with configurations with a maximum-weighted matching. The "dublin2g0" and "dublin1g0" configurations use the Gale/Shapley algorithm to compute a stable marriage while "dublin20" and "dublin10" computes a maximum-weighted matching. Both configurations use no threshold. The "dublin1g0" and "dublin10" configurations do not use extrinsic features. Figure 1 clearly shows that it is better to compute a maximum-weighted matching. This setup outperforms the stable-marriage configuration in almost all cases, sometimes drastically.

In the "dublin2e0" setup, a maximum-weighted matching is applied also in between iterations, where as for "dublin2g0" and "dublin20" the Gale/Shapley resp. the Hungarian algorithm is only applied after the last iteration. This configuration can therefore not directly be compared with "dublin2g0", but in comparison with "dublin20" it becomes clear that enforcing a one-to-one-mapping also in between iterations is better than doing so after the last iteration only.

6.3 Threshold

To find out what value for the threshold is best, we took a closer look at ontologies 205 and 303. Figure 2 shows the relation between the threshold and the precision, recall and F1 measures on ontologies 205 resp. 303. Note that varying the threshold has a quite different effect on the two ontologies. In ontology

205, recall drops faster than precision increases. The maximum F1 is reached at a threshold of 0.05. In ontology 303, precision and recall at threshold 0 are lower than in ontology 205. When raising the threshold, recall drops only slightly while precision increases rather quickly. Maximum F1 is reached at a threshold between 0.7 and 0.85. We have to conclude that the best cut-off value for our mapping algorithm depends strongly on the dataset. On the "real world" ontologies 301–304 the threshold is higher, while for the artificial benchmark ontologies the best F1 is reached at a very low threshold.

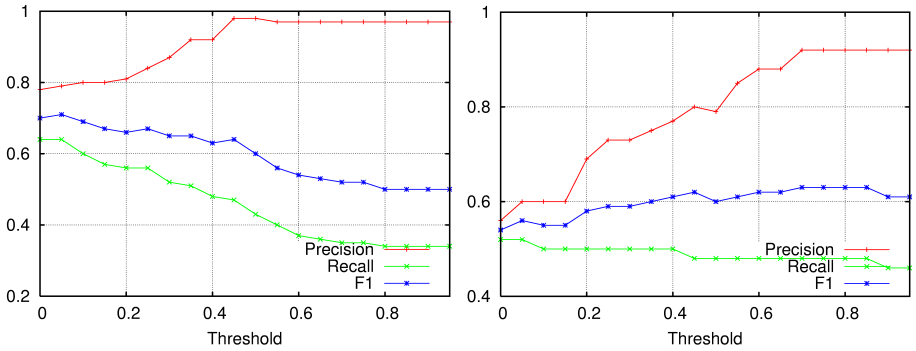

Fig. 2. Relation between threshold and precision, recall and F1 for ontologies 205 (left) and 303 (right)

6.4 Comparison with other Algorithms

To evaluate our own method, we compared our results against the published results from the 2005 Ontology Alignment Evaluation Initiative ([5]).

The algorithm that performed best in the 2005 contest was the Falcon algorithm [9] by the Southeast University of Nanjin. Falcon uses a bipartite graph representation of an RDF graph to compute structural similarity.

Our own algorithm can, however, compete well with the "FOAM" algorithm developed in Karlsruhe and the "OLA" algorithm. "Edna" is simple algorithm that is based on edit distance of the labels and was included by the organisers of the contest as a baseline. Our algorithm in the "dublin20" setting as submitted to the organisers of the OAEI 2005 performs second best after Falcon. From the 2004 algorithms, the algorithm developed at Stanford has a higher average precision, but a lower average recall than ours. These results are in greater detail presented in [5]. Figure 3 shows the F1 score. To aggregate the results of the individual tests, the organisers of the contest calculated the precision and recall over all mappings of all test.

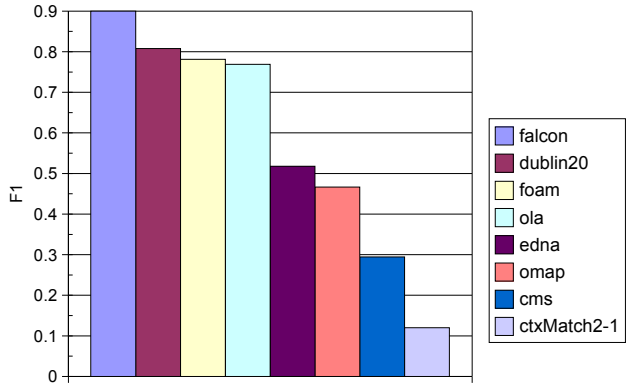

Fig. 3. Overall F1 of the OAEI 2005 alignments

6.5 Using Training Data

To test the effect of background knowledge on the performance of the alignment, we conducted experiments where we used a given mapping from the target ontology to a third (background) ontology. It is clear that we can expect the biggest improvement in accuracy if the background ontology is very similar to the source ontology. Vice versa, if the source ontology is very dissimilar, we cannot expect an improvement. Our experiments confirmed this intuition. We noticed that in the worst case using background knowledge does not improve the performance at all or could even negatively affect the performance slightly. If the right background mapping is selected, however, the performance can be increased by quite a lot. In figure 4, the experiments with training data are denoted as "amsterdam10"[8] for an experiment with intrinsic similarity only and as "amsterdam2e0" for an experiment using extrinsic similarity as well. When testing ontology 205, we used 206 as background ontology and vice versa, for 301 we used 302 as background and vice versa, and for 303 we used 304 and vice versa.

Clearly, the performance on ontologies 205 and 206 are examples for the worst case. The reason for that becomes clear if we look at the performance of the mapping from ontology 205 to 206, i.e. when mapping to the background ontology only instead of the reference ontology 101. For comparison, we include these results in the diagram and denote this configuration as "amsterdam10b " resp. "amsterdam2e0b". From these results, we can also see that the improvement of using training data in the 30x ontologies is not only due to the fact that mappings between 301 and 302 resp. 303 and 304 are easy. Rather, the combined approach of using background knowledge outperforms both the simple mapping between the source and the reference ontology and also between the source and the background ontology. We conclude that using known mappings as training

[8] This part of the research was carried out in Amsterdam, hence this name for the algorithm.

data can generally improve the results, but that the algorithm can also be misled, if the background ontology is too different from the source ontology.

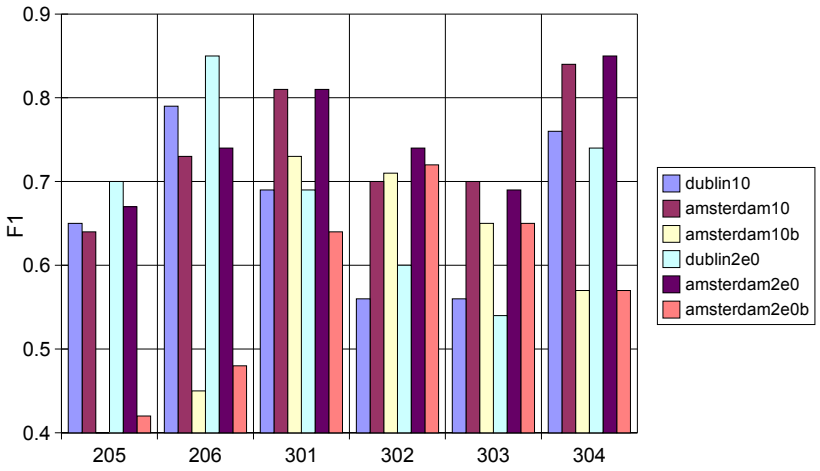

Fig. 4. Performance (F1) of setup with training data compared to normal setup

7 Conclusion

We have presented a new method for ontology mapping that uses established string distance metrics and an extrinsic feature representation as known from relational learning algorithms. We treat the results of the similarity computation as a bipartite graph and use well-known algorithms from graph theory to compute an optimal one-to-one mapping. With an empirical evaluation, we have shown that our basic ideas work, and that our algorithm can compete with other approaches. Furthermore, we have shown how our algorithm can be used in a supervised way in order to exploit background knowledge.

In the more detailed comparison in [7] we have shown that each algorithm has specific strengths and weaknesses. Therefore, we believe that there is a great potential for a combination of some of our ideas with methods used by others. We ignore some valuable information that comes from the ontologies, because we do not do any logical reasoning or inference. On the other hand, some of the methods proposed here, for example the post-processing steps, could be useful in conjunction with other base algorithms as well.

Acknowledgments. Most of the research presented in this paper was done when the author was at University College Dublin and was supported by grants from

Science Foundation Ireland and the US Office of Naval Research. The author would like to thank Nicholas Kushmerick for valuable feedback and support.

References

1. William W. Cohen, Pradeep Ravikumar, and Stephen E. Fienberg. A comparison of string distance metrics for name-matching tasks. In *Proceedings of the IJCAI-03 Workshop on Information Integration on the Web (IIWeb-03)*, pages 73–78, 2003.
2. Marc Ehrig and Steffen Staab. QOM – quick ontology mapping. In *3rd International Semantic Web Conference*, Hiroshima, Japan, 2004.
3. Jérôme Euzenat. An API for ontology alignment. In *3rd International Semantic Web Conference*, Hiroshima, Japan, 2004.
4. Jérôme Euzenat, David Loup, Mohamed Touzani, and Petko Valtchev. Ontology alignment with OLA. In York Sure, Oscar Corcho, Jérôme Euzenat, and Todd Hughes, editors, *Proceedings of the 3rd International Workshop on Evaluation of Ontology based Tools (EON)*, Hiroshima, Japan, 2004.
5. Jérôme Euzenat, Heiner Stuckenschmidt, and Mikalai Yatskevich. Introduction to the ontology alignment evaluation 2005. In *K-CAP 2005 Integrating Ontologies Workshop*, Banff, Alberta, Canada, 2005.
6. David Gale and Lloyd Stowell Shapley. College admissions and the stability of marriage. *American Mathematical Monthly*, 1962.
7. Andreas Heß. *Supervised and Unsupervised Ensemble Learning for the Semantic Web*. PhD thesis, School of Computer Science and Informatics, University College Dublin, Dublin, Ireland, 2005.
8. Andreas Heß and Nicholas Kushmerick. Iterative ensemble classification for relational data: A case study of semantic web services. In *Proceedings of the 15th European Conference on Machine Learning*, Pisa, Italy, 2004.
9. Wei Hu, Ningsheng Jian, Yuzhong Qu, and Qanbing Wang. GMO: A graph matching for ontologies. In *K-CAP 2005 Integrating Ontologies Workshop*, Banff, Alberta, Canada, 2005.
10. Vladimir I. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. *Doklady Akademii Nauk SSSR*, 163(4):845–848, 1965. In Russian. English Translation in Soviet Physics Doklady, 10(8) p. 707–710, 1966.
11. Jayant Madhavan, Philip A. Bernstein, and Erhard Rahm. Generic Schema Matching with Cupid. In *Proceedings of the 27th International Conference on Very Large Databases*, pages 129–138, Rome, Italy, 2001.
12. S. Melnik, H. Molina-Garcia, and E. Rahm. Similariy flooding: A versatile graph matching algorithm. In *Int. Conference on Data Engineering (ICDE)*, 2002.
13. James Munkres. Algorithms for the assignment and transportation problems. *SIAP*, 5(1):32–38, 1957.
14. Jennifer Neville and David Jensen. Iterative classification in relational data. In *AAAI Workshop Statistical Relational Learning*, 2000.
15. York Sure, Oscar Corcho, Jérôme Euzenat, and Todd Hughes, editors. *3rd Int. Workshop on Evaluation of Ontology based Tools (EON)*, Hiroshima, Japan, 2004.
16. William E. Winkler and Yves Thibaudeau. An application of the Fellegi-Sunter model of record linkage to the 1990 U.S. decennial census. Technical report, U.S. Bureau of the Census, Washington, D.C., 1991. Statistical Research Report Series RR91/09.