

# Simple Algorithms for Predicate Suggestions using Similarity and Co-Occurrence

Eyal Oren, Sebastian Gerke, and Stefan Decker

Digital Enterprise Research Institute  
National University of Ireland, Galway  
Galway, Ireland  
`firstname.lastname@deri.org`

**Abstract** When creating Semantic Web data, users have to make a critical choice for a vocabulary: only through shared vocabularies can meaning be established. A centralised policy prevents terminology divergence but would restrict users needlessly. As seen in collaborative tagging environments, suggestion mechanisms help terminology convergence without forcing users. We introduce two domain-independent algorithms for recommending predicates (RDF statements) about resources, based on statistical dataset analysis. The first algorithm is based on similarity between resources, the second one is based on co-occurrence of predicates. Experimental evaluation shows very promising results: a high precision with relatively high recall in linear runtime performance.

## 1 Introduction

The Semantic Web is decentralised in terms of autonomy, allowing everyone to make any statement, but centralised in terms of vocabulary: others can only understand statements that use familiar terminology. Given this situation, we consider the following problem: how to ensure that individuals, free to use arbitrary terminology, converge towards shared vocabularies?

As a particular use case we consider authoring in Semantic Wikis [12, 14, 18]. These enhanced Wikis allow users to describe information both in free text and through semantic descriptions. Allowing users to make arbitrary statements is important, since it ensures domain-independence of the Wiki.

Without further considerations, the authoring freedom in Semantic Wikis would result in statements with different vocabularies, defying the purpose of the Semantic Wiki. A terminology policy could be enforced but that would highly restrict users. A suggestion mechanism, recommending terminology based on the dataset, would help converge terminology without forcing users, as demonstrated in collaborative tagging [9, 10].

In collaborative data entry, participants construct a dataset by continuously and independently adding further statements to existing data. Each participant faces the question: *when creating Semantic Web data, which vocabulary to use?* To ensure convergence, the answer is: use the most relevant and frequently occurring vocabulary.

Finding the most *frequent* vocabulary is straightforward: one can simply count the occurrences. We therefore focus on finding the *relevant* vocabulary. Datasets typically contain heterogeneous data. Finding the vocabulary that is relevant for one resource therefore means: finding similar resources and use their vocabulary.

*Problem statement* Our problem is thus to suggest relevant and frequent terminology for extending a resource in an RDF dataset based on similarity with other resources and our question is *how well simple algorithms solve this problem?*

We present two algorithms that address this problem, based on the following hypotheses that simple algorithms do well enough: (a) computing resource similarity based only on *outgoing arcs* yields good results; (b) approximating resource similarity through *pairwise predicate co-occurrence* yields good results.

We will present the two algorithms in sections 2 and 3, and their implementation in section 4. We verify our hypotheses and the performance of these algorithms empirically in section 5. We conclude with a discussion of related work in section 6.

## 2 Classification-based algorithm

The task of the suggestion algorithm is to find, for a certain resource in focus, predicates to further describe that resource. The general idea of the classification-based algorithm is to divide the knowledge base in two groups, those similar to the current resource and those not similar, and to suggest the frequently occurring predicates from the similar group.

For example, figure 1 shows a simple knowledge base with three resources: the person “John”, with his name, some friends, and homepage, the book “The Pelican Brief”, with its title and author, and the person “Stefan”, with his name. We want to suggest relevant predicates for “Stefan” based only on the given graph.

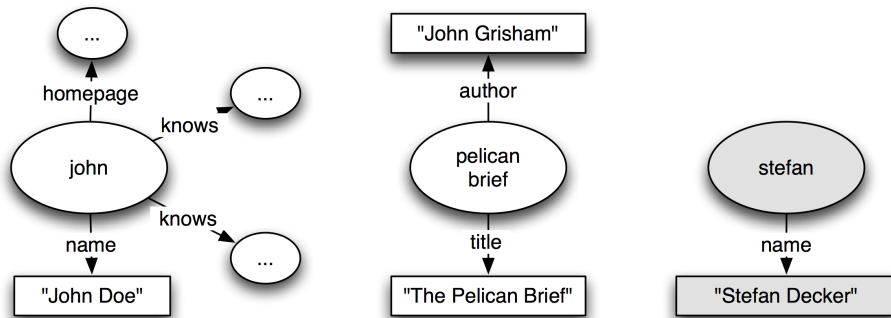


Figure 1: Example knowledge base

Listing 1.1: Classification-based algorithm

```

def suggest(r, resources)
  # select similar resources
  similar_resources = resources.select { |r'| similarity(r,r') > threshold }

  # then collect all predicates from similar resources
  candidates = similar_resources.collect { |r'| r'.predicates }

  # then rank all candidate predicates
  return rank(candidates)
end

```

The algorithm consists of two steps, as shown in listing 1.1. In the first step, we divide all existing resources in the knowledge base into two sets, the similar and unsimilar ones. In the second step, we look at all predicates from the similar group and rank them using a ranking function. In the remainder of this section, we explore each step in more detail: how to define similarity between resources, and how to rank the selected predicates.

## 2.1 Preliminaries

First we introduce some necessary definitions:

**Definition 1 (RDF Graph).** An RDF graph  $G$  is defined as  $G = (V, E, L, l)$  where  $V$  is the set of vertices (subjects and objects),  $E$  is the set of edges (predicates),  $L$  is the set of labels,  $l : E \rightarrow L$  is the labelling function for predicates. The projection  $source : E \rightarrow V$  and  $target : E \rightarrow V$  return the source and target nodes of a given edge.

**Definition 2 (Outgoing edges).** The set of outgoing edges  $E_o(v)$  of a vertex is defined as:  $E_o(v) = \{e \in E | source(e) = v\} \subseteq E$ . The bag of labels  $L(E)$  of a set of edges is defined as  $L(E) = [l(e) | e \in E]$ . The bag of labels  $L_o(v)$  of outgoing edges of a vertex  $v$  is defined as  $L_o(v) = L(E_o(v))$ . The set of outgoing edges of  $v$  whose label is  $l$  is defined as  $E_o(v, l) = \{e \in E_o(v) | l(e) = l\}$ .

## 2.2 Classification step

In the first step, we classify resources into those similar to the current one, and those not similar. The main requirement for the similarity metric is domain-independence: the algorithm should not rely on domain-specific knowledge. We use two well-known, widely used generic similarity metrics [2, 3]: the *containment* of one resource in another and their mutual *resemblance*.

Since we are interested in suggesting new predicates, we use these metrics to compare existing predicates of resources. Containment thus defines resource similarity as the amount of predicates of the first resource that are also contained in the second resource, as shown in equation (1). Resemblance measures how many of all predicates used in at least one of the two resources are used in both

resources, as shown in equation (2). For example, in figure 1, the resource “Stefan” uses the predicate “name” and the resource “John” uses “name”, “knows” and “homepage”, resulting in a containment value (of “Stefan” in “John”) of 1 and a resemblance of  $\frac{1}{3}$ .

$$s_c(v', v) = \frac{|O(v) \cap O(v')|}{|O(v)|} \quad (1)$$

$$s_r(v', v) = \frac{|O(v) \cap O(v')|}{|O(v) \cup O(v')|}. \quad (2)$$

Since predicates can have multiple values, when computing this containment or resemblance metrics we need to decide whether to count multiple predicate occurrences once or several times.

In the example, the resource “John” uses the “knows” predicate twice with different values; we can either count these two occurrences only once, thus using  $O(v)$  as a set, as shown in equation (3). The resemblance between “Stefan” and “John” would then be  $\frac{1}{3}$ . But we could also count each occurrences separately, using  $O(v)$  as a bag as shown in equation (4), yielding a resemblance of  $\frac{1}{4}$ .

$$O_s(v) = \{l(e) | e \in E_o(v)\} \quad (3)$$

$$O_b(v) = [l(e) | e \in E_o(v)] \quad (4)$$

If we generalise from these two choices, the result of the first phase is the set of similar resources  $V_s(v)$ , as defined in equation (5), where  $s_t$  is some similarity threshold and  $s(v, v')$  is either resemblance or containment measure. For example, with a threshold of 0.9 the set of similar resources to “Stefan” would consist only of the resource “John”.

$$V_s(v) = \{v' \in V : s(v, v') \geq s_t\} \quad (5)$$

### 2.3 Ranking step

After classifying all resources into two groups we collect all predicates from the set of similar resources  $V_s(v)$  and use them as candidates for the suggestion. Since there might be many candidates, we need to rank these candidates and suggest the more useful predicates first. The most straightforward ranking function is based on the occurrence frequency of these predicates in the set of similar resources.

In this example, since only the resource “John” is similar to “Stefan”, the candidates would be “knows” and “homepage”, ignoring the predicates that “Stefan” uses already. Out of these two candidates, “knows” would be ranked first since it appears most frequently.

But again, since predicates in RDF can be multi-valued, we can define the (relative) occurrence frequency of a label  $l$  in the set of similar resources  $V_s(v)$  in two ways. We can either count each predicate occurrence, as shown in equation

(6). Or we can count each occurrence only once, or stated differently, count the set  $X$  of resources that use  $l$  in their outgoing edges and divide them by the total number of resources, as shown in equations (7). In the latter case, “knows” and “homepage” would be ranked the same since they are both used by one resource.

$$r_v(e) = f_s^p(v, l) = \frac{\sum_{v' \in V_s(v)} |E_o(v', l)| \cdot w(v, v')}{\sum_{v' \in V_s(v)} |E_o(v')| \cdot w(v, v')} \quad (6)$$

$$r_v(e) = f_s^r(v, l) = \frac{\sum_{v' \in X} w(v, v')}{\sum_{v' \in V_s(v)} w(v, v')} \quad (7)$$

$$X = \{v \in V_s(v) | l \in O(v)\}$$

In both methods of counting, we could allow for a weighting factor  $w(v, v')$ . The reason for this is that even in the set of similar resources  $V_s(v)$ , some are more similar than other: in ranking the predicates, it would be natural to “promote” the predicates from similar resources over those from less similar resources. If we choose to prefer predicates from resources more similar to  $v$ , the weight factor could be given by the resource similarity, shown in equation (8). A simpler approach would not to weigh the predicates, as shown in equation (9). In our example, these methods would yield the same ranking since both candidates originate from the same resource “John”.

$$w_s(v, v') = s(v, v') \quad (8)$$

$$w_c(v, v') = \begin{cases} 1 & : v' \in V_s(v) \\ 0 & : v' \in V_n(v) \end{cases} \quad (9)$$

## 2.4 Qualitative results

To investigate our hypothesis, we have evaluated the performance and quality of the algorithm using various different datasets. We are interested in the quality of the basic algorithm (using containment, counting multi-valued predicates only once, and without weighting) and in whether the various parameters, while reducing simplicity, improve the basic algorithm. We present and discuss these results in section 5.

## 2.5 Performance

Regarding the runtime performance of the algorithm, we can analyse the description in listing 1.1. We see that, ignoring data access, the overall algorithm should run linearly to the number of resources: The first phase, classifying the similar resources, runs linear to the number of resources  $r$  and the average number of predicates per resource  $p$ : comparing the similarity of each resource against the one resource in focus by comparing all their predicates. The second phase,

ranking, is linear in the number of candidates  $c$ . The complete algorithm would therefore run in  $O(r \cdot p + c)$ , which is linear in  $r$ , since  $p$  will be constant on average and  $c$  is presumably smaller than  $r$ .

However, in practice we cannot ignore lookup performance on large datasets. To compute similarity, we need to lookup all predicates of each resource. Depending on the lookup performance of the used datastore, this could cause the whole algorithm to run logarithmic or even quadratic to the size of the dataset, rendering the algorithm impracticable for reasonably large datasets.

A simple solution would be to materialise the similarity between resources in memory, obliterating the need for data lookup during suggestion time. Direct materialisation however has two problems: the required memory space would be quadratic in the size of the dataset, and updating one resource (prone to happen often in a data entry scenario) would require recalculation of all similarity values with respect to this resource.

The next algorithm remedies exactly this problem and allows materialisation without large memory requirements.

### 3 Co-occurrence-based algorithm

The general idea of the co-occurrence-based algorithm is to approximate resource similarity through the co-occurrence of predicates. Since usually datasets contain far less predicates than resources, predicate co-occurrence requires far less space than resource similarity. We then further reduce the required space by not considering the complete power set over all predicates, but instead approximate full co-occurrence through binary co-occurrences. We thus consider only pairwise occurrences of predicates, suggest predicate candidates for each pairwise occurrence, and combine these candidates through intersection.

We therefore make two assumptions on the probabilistic model of the dataset: (1) that predicate co-occurrence correlates with resource similarity, and (2) that considering binary predicate co-occurrences to be independent events (which they are not) yields acceptable predictions. The latter allows us to pairwise consider binary co-occurrences instead of all permutations.

The algorithm is based on association rule mining [1, 17] used for recommendations in e.g. online stores: when buying one book, other books that are often bought together with this book are recommended. In our case, books are replaced by predicates and shopping transactions by resources.

#### 3.1 Precomputation step

To better show the details of the algorithm, we extend our earlier example, adding the person “Sebastian” and some more statements about John, as shown in figure 2. Again, we want to suggest further predicates to the resource “Stefan”.

In the first step we calculate usage statistics of predicates in the knowledge base. We count for each predicate, the resources that use this predicate, defined in equation (10). Secondly, we count for each pair of predicates, the number of

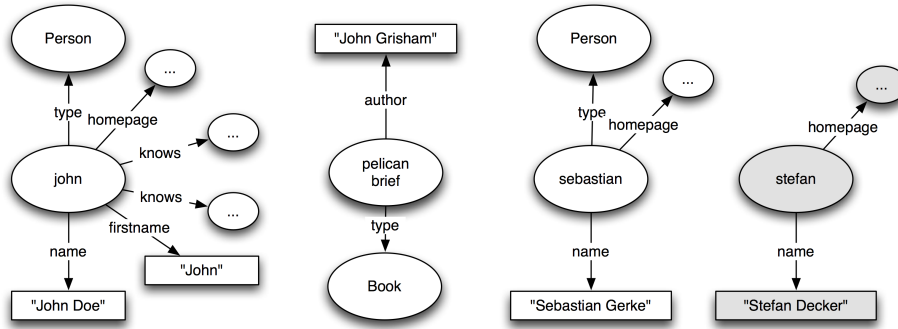


Figure 2: Extended Knowledge Base

times they co-occur together in the same resource, as defined in equation (11). The particular statistics for the example in figure 2 are given in table 1a and table 1b.

$$occ(p) = |\{v \in V | p \in L_o(v)\}| \quad (10)$$

$$coocc(p_1, p_2) = |\{v \in V | p_1 \in L_o(v) \wedge p_2 \in L_o(v)\}| \quad (11)$$

<i>predicate</i>	<i>freq.</i>		type	name	knows	homepage	firstname	author
type	3	type	3	2	1	2	1	1
name	2	name	2	2	1	2	1	0
knows	1	knows	1	1	1	1	1	0
homepage	2	homepage	2	2	1	2	1	0
firstname	1	firstname	1	1	1	1	1	0
author	1	author	1	0	0	0	0	0

(a) occurrence

(b) co-occurrence

Table 1: Predicate occurrence and co-occurrence frequency

### 3.2 Suggestion step

In the second step, we compute suggestions for a given resource. We consider all predicates in the knowledge base that occur more than once with each of the predicates from “Stefan” as suggestion candidates, as defined in equation (12).

In our example, the predicates “type”, “knows”, and “firstname” are candidates for the resource “Stefan”.

$$cooccurring(p_1) = \{p_2 : coocc(p_1, p_2) > 1\} \quad (12)$$

For each candidate we calculate our confidence in suggesting it. As shown in equation (13), the confidence for suggesting a predicate  $p$  for a selected resource  $r$ , is formed by combining the confidence for  $p$  from each of  $r$ ’s predicates  $p_i$ . In the earlier example, the total confidence for suggesting “type” is computed by combining  $confidence(name \Rightarrow type)$  and  $confidence(homepage \Rightarrow type)$ .

$$confidence(p, r) = \prod_{p_i \in cooccurring(p) \cap L_o(r)} confidence(p_i \Rightarrow p) \quad (13)$$

Each constituent is computed as shown in equation (14): the confidence for suggesting any  $p_2$  based on the existence of a  $p_1$  is given as the co-occurrence frequency of  $p_1$  and  $p_2$  relative to the occurrence frequency of  $p_1$  by itself. In our example,  $p_2$ , the candidate, would be “type”, “knows”, or “firstname”, and  $p_1$ , the existing predicates, would be “name” and “homepage”. Intuitively, we consider a relatively frequent co-occurrence as evidence for predicting  $p_2$ .

$$confidence(p_1 \Rightarrow p_2) = \frac{coocc(p_1, p_2)}{occ(p_1)} \quad (14)$$

In our example, as shown in table 2, “type” co-occurs with both predicates of “Stefan” 100% of the time, whereas the two other candidates (“knows” and “firstname”) co-occur only 50% of the time with each of the predicates of “Stefan”. We rank each candidate by the combined (unweighted) confidence: in this example, “type” will be ranked first, with a combined confidence of 100%, and the other two second, with a combined confidence of 25%.

<i>candidate</i>	<i>name</i>	<i>homepage</i>	<i>confidence</i>
type	1.0	1.0	1.0
knows	0.5	0.5	0.25
firstname	0.5	0.5	0.25

Table 2: Relative co-occurrence ratios for Stefan

## 4 Implementation

We have implemented both algorithms in Ruby. We use the ActiveRDF [13] datastore abstraction layer which allows us to run this algorithm on various RDF datastores. The implementations are distributed as part of the ActiveRDF. We

Listing 1.2: Co-occurrence as database views

```

create view occurrence as
select p, count(distinct s) as count
from triple
group by p;

create view cooccurrence as
select t0.p as p1, t1.p as p2, count(distinct t0.s) as count
from triple as t0 join triple as t1 on t0.s = t1.s and t0.p != t1.p
group by t0.p, t1.p

```

have also implemented the co-occurrence algorithm as a wrapper for an RDF datastore, in particular for the rdflite<sup>1</sup> RDF store.

Since rdflite uses a relational database with one table, `triple(s,p,o)`, we have implemented the (co)occurrence statistics as views on this database, comparable to [7]. Depending on the relational database, these views can be materialised or computed for each suggestion. The views, shown in listing 1.2, are a straightforward translation of the equations (10) and (11) given before.

#### 4.1 Example suggestions

Figure 3 shows an example of our suggestion system, for a randomly chosen resource from a dataset<sup>2</sup> about the Mindswap research group. The resource (a blank node representing Dan Connolly) and its predicates, such as name and email address, are listed on the left-hand side. Our suggestions, based on the other resources in this dataset, are listed in ranked order on the right-hand side.

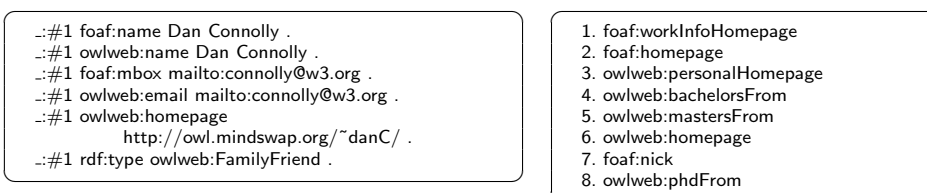


Figure 3: Suggested predicates (right) for example resource (left)

## 5 Evaluation

A predicate suggestion system is a kind of recommender system, using the opinions of a community to help individuals decide between a potentially overwhelm-

<sup>1</sup> <http://wiki.activerdf.org/rdflite/>

<sup>2</sup> <http://www.cs.umd.edu/~hendler/2003/MindPeople4-30.rdf>

ing set of choices [6, 16]. In our case, this “potentially overwhelming set of choices” is formed by the terminology (ontologies or schemas) available.

Evaluations of recommender systems can be divided into two categories [5, 6]: when regarding recommendations as an information retrieval problem (selecting the interesting predicates from all possible predicates), evaluation is usually performed off-line, focused on accuracy, and measured using precision and recall. When, on the other hand, recommendation is approached as a machine learning regression problem (learning and predicting user’s annotation preferences), evaluation is commonly performed online, focused on utility and usefulness, and measured using a training set and a test set.

### 5.1 Evaluation approach

Our evaluation combines both the information-retrieval and the machine-learning approach: we show both precision and recall ratings and evaluate our approach using training/testing datasets through a commonly applied technique of evaluating prediction of deleted values from existing data [6].

Because the distribution of data can alter the performance of the algorithms quite severely, we evaluated on five existing RDF datasets: a webcrawl<sup>3</sup> of arbitrary RDF, the Mindswap research group<sup>4</sup>, a FOAF dataset<sup>5</sup>, a terror dataset<sup>6</sup> augmented with terrorist data, and the ontoworld.org Semantic Wiki<sup>7</sup>. These datasets have differing characteristics, as shown in Table 3: both large and small, with homogeneous and heterogeneous data, and both highly structured and highly unstructured distribution.

<i>dataset</i>	<i>classes</i>	<i>resources</i>	<i>triples</i>
webcrawl	2	112	6766
mindpeople	14	273	1081
foaf	4	3123	10020
terror	25	1553	16632
ontoworld	42	4467	28593

Table 3: Evaluation datasets

Our primary evaluation technique is prediction of deleted values: we pick a random resource from the dataset as a candidate for which further predicates should be suggested. We then randomly remove one or more statements about this candidate and analyse if and at which rank position the removed predicates are re-suggested. Repeated over  $n$  random resources this yields the *average re-suggestion rate* (how often was the deleted predicate resuggested), the *empty*

<sup>3</sup> <http://www.activerdf.org/webcrawl.10k.nt>

<sup>4</sup> <http://www.cs.umd.edu/~hendler/2003/MindPeople4-30.rdf>

<sup>5</sup> <http://rdfweb.org/2003/02/28/cwm-crawler-output.rdf>

<sup>6</sup> <http://reliant.teknowledge.com/DAML/TerroristActs.owl>

<sup>7</sup> <http://ontoworld.org/RDF/>

*suggestion rate* (how often were no suggestions given), and the *average rank* of the resuggested predicate. Since in practice not all suggestions can be displayed or will be considered by the user, we also show how many of the predicates were resuggested within the top-k of suggestions.

Secondly, we measure suggestion precision (how many suggestions are valid) and recall (how many valid suggestions have we missed) based on the schema definition: we define “valid” predicates as those predicates that, according to the schema, fall within the domain of the selected candidate. For recall computation, we consider only predicates that are actually used in the dataset; since the algorithm considers only instance data, unused predicates are unattainable.

## 5.2 Results

All tests were run on an AMD Opteron 1993MHz machine with 2GB of RAM. The similarity algorithm was run 300 times over five random samples (n=100, n=150, n=200, n=250, n=300) since its performance prevented us from using the full datasets; the co-occurrence algorithm was run 20.000 times over the complete datasets. In each run, we randomly selected a resource and deleted between one and ten of its existing predicates. We then let the algorithms suggest additional predicates and compare these to the randomly deleted predicates.

We first show the results of the two primary algorithms for each dataset: table 4 shows the results of the classification-based algorithm, table 5 the results of the co-occurrence-based algorithm. The tables show, for each dataset and for all datasets combined, the resuggestion rate, empty suggestion rate and average rank. It also shows the resuggestion rate when only considering the top-k results, and the precision, recall, and the  $F_1$ -measure for each algorithm.

We can see that in general the co-occurrence performs better than any of the classification-based variants, especially when looking at the top-5 results. We can see that the co-occurrence algorithm has very high precision (100% on average). The co-occurrence algorithm has a slightly lower recall than the classification-based ones, due to the intersection of candidates which results in only high-confidence candidates. The  $F_1$ -measure (harmonic mean of precision and recall) shows that co-occurrence has the highest quality over all datasets.

<i>dataset</i>	<i>resugg.</i>	<i>empty</i>	<i>rank</i>	<i>top-5</i>	<i>top-10</i>	<i>top-20</i>	<i>prec.</i>	<i>recall</i>	$F_1$
webcrawl	0.95	0.04	1.06	0.94	0.94	0.94	0.96	0.73	0.83
mindpeople	0.80	0.19	1.30	0.79	0.80	0.80	0.81	0.83	0.83
foaf	0.92	0.06	1.30	0.92	0.93	0.93	0.94	0.80	0.87
terror	0.98	0.02	1.10	0.97	0.97	0.98	0.98	0.91	0.95
ontoworld	0.85	0.13	1.39	0.84	0.84	0.85	0.87	0.72	0.79
<i>average</i>	0.90	0.08	1.22	0.89	0.90	0.90	0.92	0.80	0.85

Table 4: Results per dataset for classification-based algorithm

<i>dataset</i>	<i>resugg.</i>	<i>empty</i>	<i>rank</i>	<i>top-5</i>	<i>top-10</i>	<i>top-20</i>	<i>prec.</i>	<i>recall</i>	<i>F<sub>1</sub></i>
webcrawl	1.00	0.00	1.18	0.99	0.99	1.00	1.00	0.74	0.85
mindpeople	1.00	0.00	1.23	1.00	1.00	1.00	1.00	0.76	0.87
foaf	1.00	0.00	1.51	0.95	1.00	1.00	1.00	0.59	0.74
terror	1.00	0.00	1.15	0.98	1.00	1.00	1.00	0.95	0.97
ontoworld	1.00	0.00	1.14	0.98	1.00	1.00	1.00	0.78	0.88
<i>average</i>	1.00	0.00	1.24	0.98	1.00	1.00	1.00	0.77	0.87

Table 5: Results per dataset for co-occurrence-based algorithm

Table 6 shows (again) the results for the primary algorithms and then lists the results for each classification variant, averaged over all five datasets. We see that using resemblance instead of containment yields very low results, which is most probably due to a too high threshold value. The other variations do not seem to affect the results much.

<i>algorithm</i>	<i>resugg.</i>	<i>empty</i>	<i>rank</i>	<i>top-5</i>	<i>top-10</i>	<i>top-20</i>	<i>prec.</i>	<i>recall</i>	<i>F<sub>1</sub></i>
co-occurrence	1.00	0.00	1.24	0.98	1.00	1.00	1.00	0.77	0.87
similarity (default)	0.90	0.08	1.22	0.89	0.90	0.90	0.92	0.80	0.85
resemblance ( $s_r$ )	0.10	0.86	1.01	0.11	0.11	0.11	0.14	0.99	0.24
similarity weigh ( $w_s$ )	0.90	0.09	1.24	0.89	0.90	0.90	0.91	0.80	0.85
count predicates ( $f_s^p$ )	0.91	0.08	1.48	0.89	0.90	0.91	0.92	0.80	0.85
threshold ( $s_t=0.8$ )	0.93	0.06	1.29	0.91	0.92	0.93	0.94	0.79	0.86

Table 6: Results of algorithm variants (averaged over all datasets)

Finally, Table 7 shows the performance times for the algorithms (only one classification variant is shown since runtime is similar for all). Figure 4 shows two graphs for these results; the left graph is zoomed for upto 300 resources, the right graph shows the full results.

Timing for the co-occurrence algorithm is divided in matrix construction and query answering. We evaluated the classification on two different datastores, rdflite and Sesame<sup>8</sup>, to evaluate scaling independent of a particular datastore implementation. We can see that the classification algorithm scales quadratic, which is due to the linear lookup times of the used datastores, although the Sesame datastore performs much better than rdflite.

Both variants of the co-occurrence algorithm perform well and scale linearly. The materialised co-occurrence implementation performs better than the view-based, which is due to the fact that the sqlite database does not support view materialisation; as mentioned earlier, both approaches have their advantages.

The classification algorithm was too slow to include tests with more than 300 resources but that was again due to data lookups on the underlying datastore: the algorithms themselves scale linearly when ignoring data-access. The materialised

<sup>8</sup> <http://www.openrdf.org>

co-occurrence implementation shows that we can circumvent data access, leading to very good performance, without requiring large memory space.

<i>algorithm</i>	<i>n=100</i>	<i>n=150</i>	<i>n=200</i>	<i>n=250</i>	<i>n=300</i>	<i>n=1555</i>	<i>n=3123</i>	<i>n=4467</i>
sim. (rdflite)	1.64s	4.02s	8.51s	15.10s	30.22s	–	–	–
sim. (Sesame)	0.71s	1.40s	2.74s	4.33s	7.88s	–	–	–
co-occ. (view)	0.63s	0.078s	1.46s	1.00s	0.93s	7.72s	9.65s	10.10s
co-occ. (constr.)	0.21s	0.27s	0.46s	0.47s	0.70s	2.73s	4.71s	6.34s
co-occ. (query)	0.01s	0.01s	0.01s	0.01s	0.01s	0.01	0.01	0.01

Table 7: Runtime performance with  $n$  resources

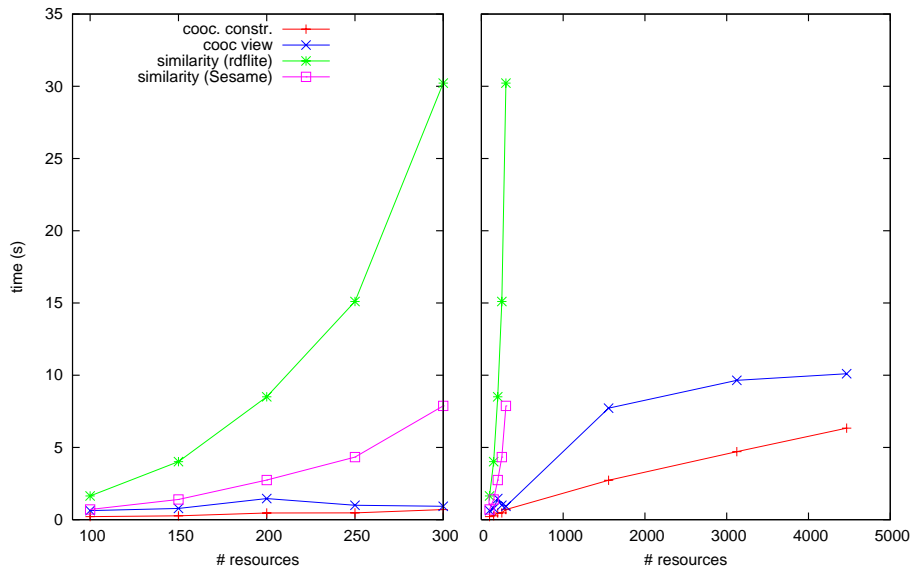


Figure 4: Runtime performance

## 6 Related Work

Annotation tools such as OntoMat [4] support semi-automatic annotation of documents; they suggest semantic annotation based on natural language analysis of the annotated resources, but do not take existing semantic descriptions into account. Annotea [8] supports collaborative annotations but annotations are made manually without a suggestion mechanism. Semantic Wikis such as

SemperWiki [14] or Semantic MediaWiki [18] allow arbitrary Semantic Web authoring but do not guide users in selecting appropriate terminology.

Automatic schema mapping techniques [15] consider a similar problem (automatically finding relations between elements of a schema) but typically operate on class-level as opposed to instance level and use e.g. concept correlation to unify schema elements [11] whereas we try to discover combined usage patterns of predicates.

Our co-occurrence algorithm is based on association rule mining [1] but our techniques for memory conservation differ: Agrawal *et al.* [1] focus on advanced pruning techniques, whereas we approximate n-ary interdependencies using pairwise binary relations (resulting in a much simplified implementation). Furthermore, our technique allows online processing with incremental updates, whereas their algorithms are iterative and need to run over the complete database.

## 7 Conclusion

We have discussed the problem of choosing vocabulary during Semantic Web data entry; a crucial bottleneck, since only through shared vocabularies can meaning be established. We introduced two algorithms for suggesting possible predicates based on statistical data analysis.

The first algorithm is based on a simple intuitive principle of resource classification: we suggest predicates from similar resources. We have discussed parametric variations that differ in the definition of similarity. We showed that the quality is good ( $F_1$  : 85%) and that variations in similarity computation do not lead to much better results.

The second algorithm approximates resource similarity through pairwise predicate co-occurrence, treating predicate occurrences as independent events (which they are not). This simplifies computation and allows for memory-efficient materialisation, while still resulting in high-quality suggestions ( $F_1$  : 87%). Runtime performance of the co-occurrence algorithm is good, scales linearly with the size of the dataset, and is constant in the presence of materialisation.

We conclude that suggesting predicates based on resource similarity works well and that, for this task, similarity based on outgoing arcs seems a “good-enough” metric. Seeing that co-occurrence suggestion quality is even better than in the classification algorithm, our second hypothesis on similarity approximation using predicate co-occurrence seems to hold as well.

*Acknowledgements* This material is based upon works supported by the Science Foundation Ireland under Grants No. SFI/02/CE1/I131 and SFI/04/BR/CS0694.

## References

- [1] R. Agrawal, T. Imielinski, and A. N. Swami. Mining association rules between sets of items in large databases. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pp. 207–216. 1993.

- [2] A. Z. Broder, S. C. Glassman, M. S. Manasse, and G. Zweig. Syntactic clustering of the web. *Computer Networks*, 29(8-13):1157–1166, 1997.
- [3] D. Dhyani, W. K. Ng, and S. S. Bhowmick. A survey of web metrics. *ACM Computer Surveys*, 34(4):469–503, 2002.
- [4] S. Handschuh. *Creating Ontology-based Metadata by Annotation for the Semantic Web*. Ph.D. thesis, University of Karlsruhe, 2005.
- [5] C. Hayes, P. Massa, P. Avesani, and P. Cunningham. An on-line evaluation framework for recommender systems. In *Workshop on Personalization and Recommendation in E-Commerce*. 2002.
- [6] J. L. Herlocker, J. A. Konstan, L. G. Terveen, and J. T. Riedl. Evaluating collaborative filtering recommender systems. *ACM Transactions on Information Systems*, 22(1):5–53, 2004.
- [7] M. Houtsma and A. Swami. Set-oriented data mining in relational databases. *Data and Knowledge Engineering*, 17(3):245–262, 1995.
- [8] J. Kahan, M. Koivunen, E. Prud’Hommeaux, and R. Swick. Annotea: An open RDF infrastructure for shared web annotations. In *Proceedings of the International World-Wide Web Conference*, pp. 623–632. 2001.
- [9] B. Lund, T. Hammond, M. Flack, and T. Hannay. A case study – Connotea. *D-Lib Magazine*, 11(4), 2005.
- [10] C. Marlow, M. Naaman, D. Boyd, and M. Davis. HT06, tagging paper, taxonomy, Flickr, academic article, to read. In *Proceedings of the ACM Conference on HyperText and Hypermedia*. 2006.
- [11] N. F. Noy and M. A. Musen. PROMPT: Algorithm and tool for automated ontology merging and alignment. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pp. 450–455. 2000.
- [12] E. Oren, J. G. Breslin, and S. Decker. How semantics make better wikis. In *Proceedings of the International World-Wide Web Conference*. 2006.
- [13] E. Oren, R. Delbru, S. Gerke, A. Haller, *et al.* ActiveRDF: Object-oriented semantic web programming. In *Proceedings of the International World-Wide Web Conference*. 2007.
- [14] E. Oren, M. Völkel, J. G. Breslin, and S. Decker. Semantic wikis for personal knowledge management. In *Proceedings of the International Conference on Database and Expert Systems Applications (DEXA)*. 2006.
- [15] E. Rahm and P. A. Bernstein. A survey of approaches to automatic schema matching. *The VLDB Journal*, 10(4):334–350, 2001.
- [16] P. Resnick and H. R. Varian. Recommender systems. *Communications of the ACM*, 40(3):56–58, 1997.
- [17] R. Srikant and R. Agrawal. Mining generalized association rules. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, pp. 407–419. 1995.
- [18] M. Völkel, M. Krötzsch, D. Vrandeovic, H. Haller, *et al.* Semantic wikipedia. In *Proceedings of the International World-Wide Web Conference*. 2006.