SCHEMA - An Algorithm for Automated Product Taxonomy Mapping in E-commerce

Steven S. Aanen, Lennart J. Nederstigt, Damir Vandić, and Flavius Frăsincar

Erasmus Universiteit Rotterdam
P.O. Box 1738, NL-3000 DR
Rotterdam, The Netherlands
{steve,lennart}@student.eur.nl, {vandic,frasincar}@ese.eur.nl

Abstract. This paper proposes SCHEMA, an algorithm for automated mapping between heterogeneous product taxonomies in the e-commerce domain. SCHEMA utilises word sense disambiguation techniques, based on the ideas from the algorithm proposed by Lesk, in combination with the semantic lexicon WordNet. For finding candidate map categories and determining the path-similarity we propose a node matching function that is based on the Levenshtein distance. The final mapping quality score is calculated using the Damerau-Levenshtein distance and a node-dissimilarity penalty. The performance of SCHEMA was tested on three real-life datasets and compared with PROMPT and the algorithm proposed by Park & Kim. It is shown that SCHEMA improves considerably on both recall and F$_1$-score, while maintaining similar precision.

Keywords: schema mapping, e-commerce, lexical matching, word sense disambiguation.

1 Introduction

In recent years the Web has increased dramatically in both size and range, playing an increasingly important role in our society and world economy. For instance, the estimated revenue for e-commerce in the USA grew from $7.4 billion in 2000 to $34.7 billion in 2007 [10]. Furthermore, a study by Zhang et al. [25] indicates that the amount of information on the Web currently doubles in size roughly every five years. This exponential growth also means that it is becoming increasingly difficult for a user to find the desired information.

To address this problem, the Semantic Web was conceived to make the Web more useful and understandable for both humans and computers, in conjunction with usage of ontologies, such as the GoodRelations [9] ontology for products. Unfortunately, as it stands today, the vast majority of the data on the Web has not been semantically annotated, resulting in search failures, as search engines do not understand the information contained in Web pages. Traditional keyword-based search cannot properly filter out irrelevant Web content, leaving it up to the user to pick out relevant information from the search results.

Search failures manifest themselves in e-commerce as well [23]. In addition, more than half of the surveyed users in the aforementioned study on online shopping in the USA [10], have encountered various frustrations when shopping online. Due to the absence of Web-wide faceted product search, it is difficult to find the product which satisfies the user's needs best. Users switch between Web-wide keyword-based search results and price comparison tools to find the 'best' product. As this is a time-consuming process, prices are often the determining factor for a purchase. This is an unwanted situation for both buyer and seller: the buyer might like a more expensive product, because it suits his needs better, whereas the seller would like to be able to differentiate his offering on other characteristics than pricing alone. The solution would be to realise a uniform presentation of Web product information, which requires the data to be annotated and structured. A method for the aggregation of data from Web stores is to use the existing hierarchical product category structure: the product taxonomy. By matching the product taxonomies from different Web stores, it becomes easier to compare their products. This should contribute towards solving the search problems encountered by users when shopping online.

In this paper we introduce the *Semantic Category Hierarchy for E-commerce Mapping Algorithm* (SCHEMA), to be used for mapping between heterogeneous product taxonomies from multiple sources. It employs *word sense disambiguation* techniques, using WordNet [17], to find synonyms of the correct sense for the category name. Furthermore, it uses lexical similarity measures, such as the *Levenshtein distance* [14], together with structural information, to determine the best candidate category to map to. In order to evaluate SCHEMA, its performance is compared on recall and precision with PROMPT [19] and the algorithm proposed by Park & Kim [20].

The structure of the paper is as follows. In Sect. 2 related work is reviewed. Section 3 presents SCHEMA, our framework for taxonomy mapping. Section 4 discusses the evaluation results of SCHEMA, compared to existing approaches. Last, in Sect. 5, we give our conclusions and suggest future work.

2 Related Work

The field of taxonomy or schema mapping has generated quite some interest in recent years. It is closely related to the field of ontology mapping, with one important difference: whereas for matching of taxonomies (hierarchical structures), and schemas (graph structures), techniques are used that try to guess the meaning implicitly encoded in the data representation, ontology mapping algorithms try to exploit knowledge that is explicitly encoded in the ontologies [22]. In other words, due to the explicit formal specification of concepts and relations in an ontology, the computer does not need to guess the meaning. In order to interpret the meaning of concepts in an ontology or schema, algorithms often exploit the knowledge contained in generalised *upper ontologies*, such as SUMO [18] or WordNet [17]. In this way the semantic interoperability between different ontologies is enhanced, facilitating correct matching between them. The semantic lexicon WordNet plays a specifically important role in many mapping algorithms, helping to overcome the ambiguity occurring in natural language, often in combination with word sense disambiguation approaches, such as the approach of Lesk [2,13]. In addition to the usage of upper ontologies for producing the mappings between ontologies and schemas, lexical similarity measures are also often used. Using lexical similarity measures helps algorithms to deal with slight lexical variations in words. The Levenshtein distance [14] is known as the edit distance, and has been augmented to allow for transposition of characters in the Damerau-Levenshtein distance [4], both utilised in our algorithm.

In their algorithm for product taxonomy mapping, Park & Kim [20] propose to use a disambiguation technique in combination with WordNet to obtain synonyms for a category name, in order to find candidate paths for matching. The candidate paths are assessed using co-occurrence and order consistency, which evaluate the overlap and the order of the categories between the source and candidate path, respectively. While specifically targeted at e-commerce, some phenomenons that occur frequently in product taxonomies are neglected, such as composite categories, in which multiple concepts are combined. Various other (database) schema matching approaches exist. SimilarityFlooding [16] uses the similarity between adjacent elements of schema entities to score possible mappings, but does not take the frequently occurring terminological variations, applicable to e-commerce, into account. COMA++ [1] provides a collection of simple matching algorithms and combinations of these. Some approaches use class attribute data for matching, such as S-Match [8] and CUPID [15]. A good overview of existing approaches has been made in recent surveys for schema matching [5,21,22].

PROMPT [19] is a general-purpose ontology mapping tool, which uses predefined (automatic or manual) mappings, called *anchors*, as guidelines for the mapping of similar nodes. However, due to its emphasis on mapping ontologies in general, it fails in matching many categories when employed for product taxonomy mapping. H-Match [3] uses WordNet for determining the correct contextual and linguistic interpretation of concepts, combined with semantic ontology data. Yu et al. [24] propose to use an upper ontology, in order to create a semantic bridge between various e-commerce standards. QOM [7] uses only simple similarity measures, aiming to reduce time complexity, without significant loss of accuracy. Ehrig & Sure [6] propose a rule-based approach, combined with neural networks. Other approaches are discussed in recent surveys for ontology mapping [11].

3 SCHEMA

This section discusses the SCHEMA framework, together with all the assumptions for our product taxonomy matching algorithm. Figure 1 illustrates the high-level overview of the framework. This sequence of steps is executed for every category in the source taxonomy. First, the name of the source category is disambiguated, to acquire a set of synonyms of the correct sense. This set is used to find candidate categories from the target taxonomy, and is needed to account for the varying denominations throughout taxonomies. After the *Candidate Target Category Selection*, every candidate category path is compared with the path of the source category, by means of the *Candidate Target Path Key Comparison*. The best-fitting candidate target category is selected as the winner. The objective of SCHEMA is to map source categories to a selected target category, if and only if, all products in the source category fit in the selected target category. This reflects our definition of a successful and meaningful category mapping. First, the general assumptions — the basis for the development of SCHEMA — are explained. Next, each step of the framework, as shown in Fig. 1, will be discussed in more detail.

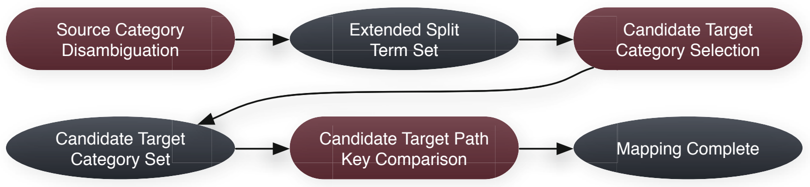
Source Category Disambiguation
Extended Split Term Set
Candidate Target Category Selection
Candidate Target Category Set
Candidate Target Path Key Comparison
Mapping Complete

Fig. 1. Framework overview for SCHEMA

3.1 General Assumptions

In product taxonomies, a frequently seen phenomenon is that of composite categories. These are nodes, that combine multiple — usually related — classes into one, like category 'Movies, Music & Games' from Amazon. Each of the three parts could have been a separate class as well, as different product concepts are represented. An assumption in the development of SCHEMA was that composite categories need to be treated adequately, as the target taxonomy might not use the same conjunction of classes. To handle the phenomenon, SCHEMA splits categories on ampersands, commas, and the string 'and'. The resulting set of classes, making up the composite category, is called the *Split Term Set*.

Product taxonomies are tree-structured data schemes, and thus have a root node. However, in product taxonomies, root categories (e.g. 'Products' or 'Shopping') are meaningless, as they do not provide information about the products falling under. The assumption used for SCHEMA is that, as root nodes are meaningless, they should get automatically mapped in taxonomy matching. Furthermore, roots should be disregarded in all computations, such as in path comparisons. Fig. 2 shows that the root categories in dark blue (the left-hand side categories in black & white printing) are matched by SCHEMA, despite being lexically dissimilar.

Between different product taxonomies, it is apparent that varying degrees of specialisation exist with respect to the product classification. This could mean

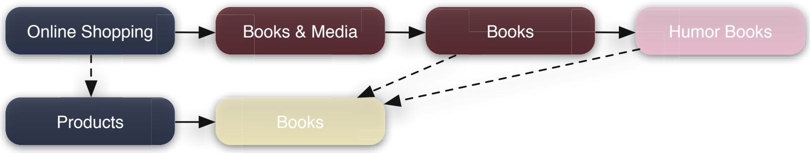

Fig. 2. Mapping example for Overstock (top) to Amazon (bottom) categories. Normal lines indicate a parent-child relationship; dashed lines indicate SCHEMA's mapping.

that there possibly is no direct match for a very specific source category in the target taxonomy. In such a case, it makes sense to match the source category to a more general target category, as from a hierarchical definition, products from a specific category should also fit into a more general class. Figure 2 shows that category 'Books' (Overstock) is mapped to 'Books' (Amazon), as one would expect. Unfortunately, there is no direct match for 'Humor Books' (Overstock) in Amazon. However, humor books are also a kind of books, so SCHEMA will map this category to the more general 'Books' category from Amazon. The more general category is found by following the defined mapping for the parent of the current source category. Note that root mappings are precluded.

SCHEMA's last assumption is, that as usage of capitals in category names does not affect the meaning, all lexical matching is performed case-insensitive.

3.2 Source Category Disambiguation

The first step in creating a mapping for a category from the source taxonomy, is to disambiguate the meaning of its name. As different taxonomies use varying denominations to identify the same classes, it is required that synonyms of the source category label are taken into account for finding candidate target categories. However, using all synonyms could result in inclusion of synonyms of a faulty sense, which could for example cause a 'laptop' to be matched with a book to write notes (i.e., a notebook). To account for this threat, SCHEMA uses a disambiguation procedure in combination with WordNet [17], to find only synonyms of the correct sense for the current source category. This procedure is based on context information in the taxonomy, of which can be expected that it gives some insight into the meaning of the source category name. Concerning the general assumption on composite categories in Sect. 3.1, SCHEMA disambiguates every part of the source category (Split Term Set) separately. The result after disambiguation is called the *Extended Split Term Set*. Note that the target taxonomy does not play a role in the source category disambiguation.

Algorithm 1 explains the procedure that is used to create the Extended Split Term Set for the current source category. First, Algorithm 1 splits the (composited) source category into separate classes: the Split Term Set. The same split is performed for all children, and for the parent of the source category, which will act as 'context' for the disambiguation process. Next, the disambiguation procedure itself, which will be discussed shortly, is called for every split part of the source category. The result, the Extended Split Term Set, contains a set of synonyms of the correct sense for each individual split term. The Extended Split Term Set is used in SCHEMA to find candidate target categories, and to evaluate co-occurrence of nodes for path-comparison.

Algorithm 1. Finding Source Category's Extended Split Term Set

```
Require: source category to disambiguate: $w_{\text{category}}$
Require: source category's parent: $w_{\text{parent}}$, and set of its children: $W_{\text{children}}$
Require: function splitComposite($w$), which splits composite category name $w$ into a
    set of individual classes: a split term set $W$
Require: function disambiguate($w_{\text{target}}, W_{\text{context}}$), disambiguates a word using a set
    of context words, resulting in a set of correct synonyms (described by Algorithm 2)
 1: {First, all used categories get split on composite classes}
 2: $W_{\text{category}} \leftarrow$ splitComposite($w_{\text{category}}$)
 3: $W_{\text{parent}} \leftarrow$ splitComposite($w_{\text{parent}}$)
 4: $W_{\text{child}} \leftarrow \emptyset$
 5: for all $w_{\text{currentChild}} \in W_{\text{children}}$ do
 6:     $W_{\text{child}} \leftarrow W_{\text{child}} \cup$ splitComposite($w_{\text{currentChild}}$)
 7: end for
 8: $W_{\text{context}} \leftarrow W_{\text{child}} \cup W_{\text{parent}}$
 9: $extendedSplitTermSet \leftarrow \emptyset$
10: {For every split part of the source category, find the extended term set}
11: for all $w_{\text{srcSplit}} \in W_{\text{category}}$ do
12:     $extendedTermSet \leftarrow$ disambiguate($w_{\text{srcSplit}}, W_{\text{context}}$)
        {Always include original split term, also when WSD is unsuccesful}
13:     $extendedTermSet \leftarrow extendedTermSet \cup \{w_{\text{srcSplit}}\}$
14:     $extendedSplitTermSet \leftarrow extendedSplitTermSet \cup \{extendedTermSet\}$
15: end for
16: return $extendedSplitTermSet$
```

As explained before, disambiguation of the source category name is based on a set of words from its context. The idea to use this context is based on a well-known algorithm for word sense disambiguation from Lesk [13]. However, traditional dictionary glosses, used by Lesk, may not provide sufficient vocabulary for successful matching. Therefore Banerjee & Pedersen [2] propose to use the rich semantic relations of WordNet, considering also related glosses of both target and context words to reduce this effect. Unfortunately, this introduces another problem: the computation time increases exponentially with the number of context words. To prevent computation time from exploding, Kilgarriff & Rosenzweig [12] propose to use Lesk's traditional algorithm with heuristics to simplify the search. Instead of using a dictionary gloss for every context word, they propose to use only the context words. This method reduces time complexity, but has similar vocabulary-related restrictions as the original Lesk algorithm. SCHEMA uses the best of these procedures, utilising the rich semantic relations of WordNet for the target word, while comparing only to the plain terms from the context, as described in Algorithm 2. For every possible sense of the target word, the overlap between its related glosses and the plain context words is assessed. The length of the longest common substring is used as similarity measure, and the sense with the highest accumulated score is picked as winner.

Algorithm 2. Context-Based Target Word Disambiguation

```
Require: word to disambiguate: $w_{\text{target}}$, and set of context words: $W_{\text{context}}$
Require: function getSynsets($w$), gives all synonym sets (representing one sense in
    WordNet), of which word $w$ is a member
Require: function getRelated($S$), gives synonym sets directly related to synset $S$ in
    WordNet, based on hypernymy, hyponymy, meronymy and holonymy. Result in-
    cludes synset $S$ as well.
Require: function longestCommonSubstring($w_a, w_b$), which computes the length of
    the longest common sequence of consecutive characters between two strings, cor-
    rected for length of the longest string, resulting in an index in the range $[0, 1]$
Require: function getGloss($S$), returns the gloss associated to a synset $S$ in WordNet
 1: $Z \leftarrow$ getSynsets($w_{\text{target}}$) {$Z$ holds all possible senses}
 2: $bestScore \leftarrow 0$
 3: $bestSynset \leftarrow \emptyset$
    {Evaluate every possible sense (synset) $S \in Z$ of target word $w_{\text{target}}$}
 4: for all $S \in Z$ do
 5:    $senseScore \leftarrow 0$
 6:    $R \leftarrow$ getRelated($S$)
       {For every combination of context words & (related) glosses, check similarity}
 7:    for all $(S_{\text{related}}, w_{\text{context}}) \in R \times W_{\text{context}}$ do
 8:       $gloss \leftarrow$ getGloss($S_{\text{related}}$)
 9:       $senseScore \leftarrow senseScore +$ longestCommonSubstring($gloss, w_{\text{context}}$)
10:    end for
11:    if $senseScore > bestScore$ then
12:       $bestScore \leftarrow senseScore$
13:       $bestSynset \leftarrow S$ {Update best known synset so far}
14:    end if
15: end for
16: return $bestSynset$
```

3.3 Candidate Target Category Selection

The result of the Source Category Disambiguation, the Extended Split Term Set, is used to find matching categories in the target taxonomy. This set of candidate categories is basically a pre-selection for the decision to which target category the current category can be mapped to. The selection relies on SCHEMA's definition of a category node match, *Semantic Match*, described by Algorithm 3, which is used consistently throughout SCHEMA. It is used to classify a source category and a target category as equivalent or dissimilar, utilising the enriched information provided by the Extended Split Term Set for the source category, in combination with lexical matching to evaluate similarity between the category names. For the composite categories, SCHEMA assumes that with respect to the split terms, the source category is a subset of the target category. This ensures that all products in a mapped source category fit in the target category.

For every split part of the source category, Semantic Match checks whether there is a matching part in the target category. A match can mean either that the source split part is contained as separate component in a target part, or that they share a lexical similarity based on the normalised Levenshtein index [14], exceeding a chosen threshold. When all split parts of the source category have a match in the target category, the match is considered semantically correct.

Algorithm 3. Semantic Match

```
Require: extended split term set $E$, with sets of synonyms $S$ of the correct sense for
    every split term of the source category
Require: target taxonomy category name: $w_{\text{target}}$
Require: Node Match Threshold $t_{\text{node}}$, defines the minimum degree of lexical simi-
    larity in order to classify two class names as equal
Require: function splitComposite($w$), splits composite category name $w$ into a set of
    individual classes: a split term set $W$
Require: function levenshtein($w_a, w_b$), computes the edit distance between two strings
Require: function containsAsSeparateComponent($w_a, w_b$), indicates whether string
    $w_a$ contains string $w_b$ as separate part (middle of another word is not sufficient)
 1: $W_{\text{target}} \leftarrow$ splitComposite($w_{\text{target}}$)
 2: $subSetOf \leftarrow$ true {Starting assumption: source split term set is subset of target}
 3: for all $S_{\text{srcSplit}} \in E$ do
 4:    $matchFound \leftarrow$ false
 5:    for all $(w_{\text{srcSplitSyn}}, w_{\text{targetSplit}}) \in S_{\text{srcSplit}} \times W_{\text{target}}$ do
 6:       $edit\_dist \leftarrow levenshtein(w_{\text{srcSplitSyn}}, w_{\text{targetSplit}})$
          {Normalise distance based on length and convert to similarity measure}
 7:       $similarity \leftarrow 1 - edit\_dist / \max(w_{\text{srcSplitSyn}}, w_{\text{targetSplit}})$
 8:       if containsAsSeparateComponent($w_{\text{targetSplit}}, w_{\text{srcSplitSyn}}$) then
 9:          $matchFound \leftarrow$ true
10:       else if $similarity \geq t_{\text{node}}$ then
11:          $matchFound \leftarrow$ true
12:       end if
13:    end for
14:    if $matchFound =$ false then
15:       $subSetOf \leftarrow$ false
16:    end if
17: end for
18: return $subSetOf$
```

Figure 3 shows some candidates that have been found for category ‘Tubs’ from Overstock. The Source Category Disambiguation procedure discussed in Sect. 3.2 results in the following Extended Split Term Set: {{`Tubs, bathtub, bathing tub, bath, tub`}}. Synonym ‘bath’ is sufficient for candidate category ‘Kitchen

& Bath Fixtures' (at the top of Fig. 3), to be selected. As 'bath' is included in split target part 'Bath Fixtures' (as separate word), it matches, according to Algorithm 3, making target category 'Kitchen & Bath Fixtures' a superset of source category 'Tubs'. Hence it is classified as a semantic match, and thus selected as proper candidate target category.

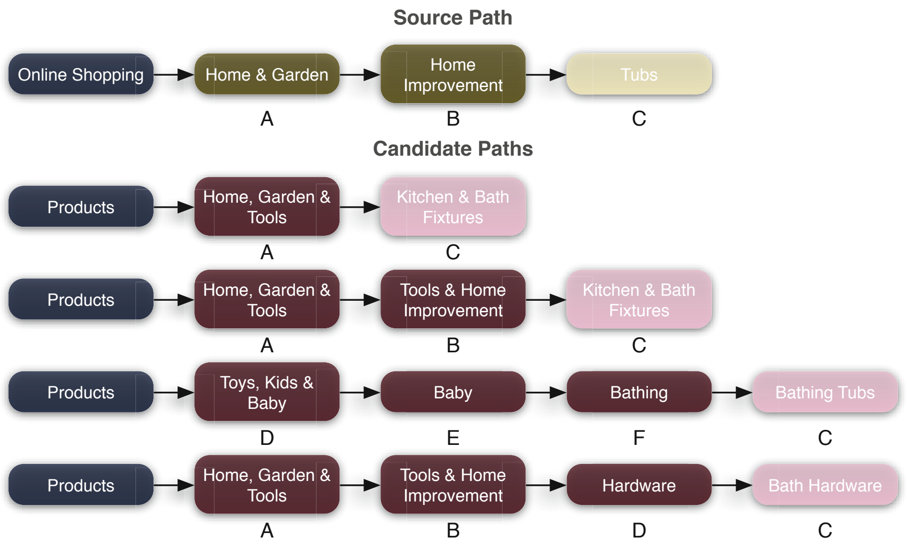

Fig. 3. Source category path for 'Tubs' in Overstock, with associated candidate target categories from Amazon

3.4 Candidate Target Path Key Comparison

SCHEMA's last step is to select the best alternative from the set of found candidate target categories, using a method that scores the similarity of the source category path against a candidate target path. This *Candidate Target Path Key Comparison* is used for every element from the set of candidate target paths. The candidate with the highest score is selected as winner. The idea of the Candidate Path Key Comparison is simple in nature, though powerful in the sense that it assesses similarity based on both structural and lexical relatedness.

For both source and candidate target path, a key is generated for every node (category) in the path. This is done in such a way, that every unique node gets a unique key. Similarly, when two nodes — independent from the path they come from — are seen as identical, they are labelled with the same key. An important question is: when are two nodes seen as identical? A straightforward way would be to base this purely on lexical similarity of the category names. However, SCHEMA uses a richer source of information for nodes from the source path: the Extended Split Term Set. Two nodes from the source path are seen as identical if and only if their Extended Split Term Sets are the same. A node from the source path and a node from the candidate target path are seen as identical when Algorithm 3, the Semantic Match procedure, decides so. The result is a key list for both the source path and the current candidate target path.

Figure 3 shows the key list for the source and candidate targets paths for category 'Tubs'. The candidate path at the bottom, is a good example of how Semantic Match classifies nodes as being similar. Candidate node 'Tools & Home Improvement' is assigned the same key ('B') as source node 'Home Improvement', as the first one is a superset of the last one, thus all products under the second should fit into the first. Considering candidate 'Bath Hardware' itself, one of the synonyms of source category 'Tubs' ('bath'), is included in the name of the candidate category. Hence, 'Bath Hardware' gets the same key ('C') as 'Tubs'.

For the key lists found for source and candidate path, the similarity is assessed using the Damerau-Levenshtein distance [4]. This measure captures the (dis)similarity and transposition of the nodes, hence both the number of co-occurring nodes and the consistency of the node order are taken into account. As the Damerau-Levenshtein distance is used in normalised form, a dissimilar node in a long candidate path is weighted as less bad than the same dissimilar node in a shorter path, which can unfortunately lead to biased results. Therefore, a penalty is added for every unique key assigned solely to the candidate path, or more precise: for every node for which no match exists in the source path. The formula used as similarity measure for the key lists is as follows:

$$candidateScore = 1 - \frac{\text{damLev}(K_{\text{src}}, K_{\text{candidate}}) + p}{\max(K_{\text{src}}, K_{\text{candidate}}) + p} \tag{1}$$

where K is a key list, p the penalty (# dissimilar nodes in candidate path), damLev() computes the Damerau-Levenshtein distance between two key lists, and max() computes the maximum length of two key lists.

In Fig. 3, the uppermost and lowermost candidate paths give an example of the penalty's usefulness. One is too short, the other too long. The shortest ('Kitchen & Bath Fixtures') does not contain new nodes in comparison to the source path. With just one edit operation (insertion of key 'B'), it gets a candidate score of $1 - \frac{1+0}{3+0} = \frac{2}{3}$. The longest contains a new node: 'Hardware'. This gives the long path a penalty of 1, while the edit distance is also 1 (deletion of key 'D'), resulting in a score of $1 - \frac{1+1}{4+1} = \frac{3}{5}$. Without penalty it would score $\frac{3}{4}$, causing it to win from the short path, which does not contain unrelated nodes. Clearly, we prefer the first candidate path, because the second candidate path possibly changes the meaning of node 'C' as it has as parent a new node 'D'.

Once the candidate target category with the highest score has been found, it is mapped if the score exceeds the *Final Similarity Threshold* (t_{final}). This threshold prevents the algorithm of performing incorrect mappings, and should not be confused with the Node Match Threshold used in Algorithm 3. When a path does not pass the Final Similarity Threshold, or when no candidate paths have been found, the source category is mapped to the mapping of its parent (but excluding the root), according to the assumption in Sect. 3.1. The complete framework procedure then repeats for the next source taxonomy category.

4 Evaluation

In order to assess SCHEMA's performance, it is compared to similar algorithms. We have chosen to compare it with PROMPT [19], being a general-purpose algorithm that is well-known in the field of ontology mapping. Additionally, the algorithm of Park & Kim [20] is included in the comparison, due to their focus on product taxonomy mapping in particular. First, we briefly discuss how the evaluation has been set up. Then, we present the results for each algorithm and discuss their relative performance.

4.1 Evaluation Design

Three product taxonomies from real-life datasets were used for the evaluation. The first dataset contains more than 2,500 categories and is from Amazon (`www.amazon.com`). The second dataset contains more than 1,000 categories and is from Overstock (`www.o.co`). Overstock is an online retailer with RDFa-tagged product pages for the GoodRelations [9] ontology. The last dataset contains over 44,000 categories and is from the shopping division in the Open Directory Project (ODP, `www.dmoz.org`). Using these three datasets, six different combinations of source and target taxonomies can be made. In order to evaluate the algorithms' performance on the mappings, it is required that each of the mappings is done manually as well. However, as the datasets are too large to manually map every category, we have taken a random sample of five hundred category nodes from each dataset. For every node it is assured that its ancestors are included in the sample as well. The mappings are made from a sampled source taxonomy to a full target taxonomy. Occasionally there are multiple nodes in the reference taxonomy to which a source category node could be correctly mapped. To account for this fact, the manual mapping may define multiple correct mappings for each source category node. The manual mappings were collectively made by three independent individuals, in order to prevent bias.

Each algorithm performed a mapping for every combination of datasets. SCHEMA and the algorithm of Park & Kim carried out multiple mappings, with different parameter values for each combination. Both algorithms use a final score threshold, referred to as t_{final}, ranging from 0 to 1, with increments of 0.05. Furthermore, SCHEMA uses a threshold for node matching, denoted by t_{node}, with range 0.50 to 1 and increments of 0.025. The completed mappings, generated by the algorithms, are compared with the manual mappings, in order to obtain their performance measures. Though ordinary classification and confusion matrix measures apply, the situation is slightly different as there are n 'positive' classes (all target categories), and only one negative (null mapping). We therefore define the 'false positives' as number of mappings to an incorrect path (either wrong or null), and the 'false negative' as incorrect mappings to null. The 'true' classes are similar to those in binary classification.

4.2 Results

Table 1 presents a comparison of average precision, recall and F_1-score for every algorithm. Tables 2, 3, and 4 give a more detailed overview of the results achieved by SCHEMA, the algorithm of Park & Kim, and PROMPT, respectively.

Table 1. Comparison of the best average results for each algorithm

Algorithm	Precision	Recall	F_1-score	Senses found	WSD accuracy
PROMPT	28.93%	16.69%	20.75%	n/a	n/a
Park & Kim	47.77%	25.19%	32.52%	5.70%	83.72%
SCHEMA	42.21%	80.73%	55.10%	82.03%	84.01%

Table 2. Best results for SCHEMA

Mapping	Precision	Accuracy	Specificity	Recall	F_1-score	t_{node}	t_{final}
A → ODP	27.27%	40.00%	34.12%	52.50%	35.90%	0.800	0.25
A → O.co	36.34%	49.40%	34.30%	82.69%	50.49%	0.850	0.15
ODP → A	57.49%	68.94%	51.70%	93.66%	71.24%	0.875	0.30
ODP → O.co	39.13%	50.70%	29.59%	95.03%	55.43%	0.850	0.25
O.co → A	53.72%	56.60%	29.13%	84.96%	65.83%	0.850	0.15
O.co → ODP	39.30%	45.80%	27.27%	75.52%	51.69%	0.925	0.30
Average	42.21%	51.91%	38.26%	80.73%	55.10%		

As shown in Table 1, SCHEMA performs better than PROMPT and the algorithm of Park & Kim, on both average recall and F_1-score. The recall has improved considerably with 221% in comparison to the algorithm from Park & Kim, and 384% against PROMPT. This can be partly attributed to the ability of SCHEMA to cope with lexical variations in category names, using the Levenshtein distance metric, as well as the ability to properly deal with composite categories. Furthermore, SCHEMA maps a category node to its parent's mapping when no suitable candidate path was found, improving the recall when the reference taxonomy only includes a more general product concept. Achieving a high recall is important in e-commerce applications, as the main objective is to automatically combine the products of heterogeneous product taxonomies in one overview, in order to reduce search failures. A low recall means that many categories would not be aligned, which would mean that many products will be missing from search results. For this reason, it is generally better to map to a more general category rather than not mapping at all. Worthy to mention is the slight decrease in average precision for SCHEMA compared with the algorithm of Park & Kim: 42.21% against 47.77%. This is due to the fact that there is a trade-off between precision and recall: achieving a higher recall means that an

Table 3. Best results for Park & Kim algorithm

Mapping	Precision	Accuracy	Specificity	Recall	F_1-score	t_{final}
A → ODP	35.77%	34.00%	57.89%	16.84%	22.90%	0.05
A → O.co	60.16%	47.20%	76.78%	25.61%	35.92%	0.00
ODP → A	37.06%	41.48%	51.94%	30.29%	33.33%	0.00
ODP → O.co	36.76%	35.87%	48.68%	25.09%	29.82%	0.10
O.co → A	61.14%	36.20%	52.11%	29.89%	40.15%	0.00
O.co → ODP	55.71%	36.60%	62.87%	23.42%	32.98%	0.50
Average	47.77%	38.56%	58.38%	25.19%	32.52%	

Table 4. Best results for PROMPT

Mapping	Precision	Accuracy	Specificity	Recall	F_1-score
A → ODP	13.55%	25.40%	44.17%	8.08%	10.12%
A → O.co	51.69%	45.40%	74.44%	22.02%	30.89%
ODP → A	20.20%	35.47%	46.44%	19.61%	19.90%
ODP → O.co	20.86%	29.86%	42.64%	16.18%	18.22%
O.co → A	50.00%	32.20%	45.96%	25.66%	33.92%
O.co → ODP	17.27%	25.80%	47.73%	8.57%	11.46%
Average	28.93%	32.36%	50.23%	16.69%	20.75%

algorithm has to map more categories, resulting in possible imprecision when the similarity between categories is low. Both SCHEMA and the algorithm of Park & Kim use configurable final thresholds to filter out weaker matches, but it cannot fully prevent mistakes from occurring. Despite the slightly worse performance on precision, SCHEMA manages to find a more suitable trade-off between precision and recall for product taxonomy mapping than PROMPT and the algorithm of Park & Kim. This is illustrated by the good performance on recall and the higher F_1-score of 55.10%. PROMPT uses a conservative mapping approach, well-suited for general ontology mapping, but unsuitable for e-commerce due to the small portion of mappings. The algorithm of Park & Kim performs better in this regard, especially on precision, but the recall is hampered by the fact that it neglects the existence of composite categories. Furthermore, it uses a rather strict lexical matching procedure between category names, in which a category name has to be a full substring of the other, creating issues when slight lexical variations occur. In addition, the disambiguation procedure from Park & Kim only manages to find a sense in WordNet in 5.70% of the total categories on average. Unfortunately, the rather good accuracy of disambiguation (83.72%) is therefore based on a very small amount of cases, making the number rather untrustworthy. The Lesk-based disambiguation algorithm employed by SCHEMA performs well on both the percentage of senses found and the accuracy, scoring 82.03% and 84.01%, respectively.

5 Conclusions and Future Work

This paper proposes SCHEMA, an algorithm capable of performing automated mapping between heterogeneous product taxonomies in e-commerce. The main objective for developing SCHEMA is facilitating the aggregation of product information from different sources, thus reducing search failures when shopping online. To achieve this objective, SCHEMA utilises word sense disambiguation techniques on category labels, based on the ideas from the algorithm proposed by Lesk [13], in combination with the WordNet semantic lexicon. Furthermore, it deals with domain-specific characteristics, such as composite categories, and lexical variations in category labels. It employs a node matching function, based on inclusiveness of the categories in conjunction with the Levenshtein distance for the class labels, for finding candidate map categories and for assessing the path-similarity. The final mapping quality score is calculated using the Damerau-Levenshtein distance, with an added penalty for dissimilar nodes in the target category's path.

The performance of our algorithm was tested on three real-life datasets and compared with the performance of PROMPT and the algorithm of Park & Kim. This evaluation demonstrates that SCHEMA achieves a considerably higher average recall than the other algorithms, with a relatively small loss of precision. The average F_1-score resulted in 55.10% for SCHEMA, against 20.75% for PROMPT, and 32.52% for the algorithm of Park & Kim.

As future work, we would like to improve SCHEMA by making use of part-of-speech tagging. As a noun is often more important for concept similarity than an adjective, it makes sense to distinguish between them and treat them accordingly. Another possibility is to combine the hierarchical category structure with product information, as the data fields in product instances could yield extra information for the taxonomy mapping. Additionally, this work could support the creation of an automatic product comparison Web site, capable of autonomously matching products and product taxonomies from different sources.

References

1. Aumueller, D., Do, H.H., Massmann, S., Rahm, E.: Schema and Ontology Matching with COMA++. In: ACM SIGMOD International Conference on Management of Data 2005 (SIGMOD 2005), pp. 906–908. ACM (2005)
2. Banerjee, S., Pedersen, T.: An Adapted Lesk Algorithm for Word Sense Disambiguation Using WordNet. In: Gelbukh, A. (ed.) CICLing 2002. LNCS, vol. 2276, pp. 136–145. Springer, Heidelberg (2002)
3. Castano, S., Ferrara, A., Montanelli, S.: H-MATCH: An Algorithm for Dynamically Matching Ontologies in Peer-Based Systems. In: 1st VLDB Int. Workshop on Semantic Web and Databases (SWDB 2003), pp. 231–250 (2003)
4. Damerau, F.J.: A Technique for Computer Detection and Correction of Spelling Errors. Communications of the ACM 7(3), 171–176 (1964)
5. Do, H.-H., Melnik, S., Rahm, E.: Comparison of Schema Matching Evaluations. In: Chaudhri, A.B., Jeckle, M., Rahm, E., Unland, R. (eds.) Web Databases and Web Services 2002. LNCS, vol. 2593, pp. 221–237. Springer, Heidelberg (2003)

6. Ehrig, M., Sure, Y.: Ontology Mapping - An Integrated Approach. In: Bussler, C.J., Davies, J., Fensel, D., Studer, R. (eds.) ESWS 2004. LNCS, vol. 3053, pp. 76–91. Springer, Heidelberg (2004)
7. Ehrig, M., Staab, S.: QOM – Quick Ontology Mapping. In: McIlraith, S.A., Plexousakis, D., van Harmelen, F. (eds.) ISWC 2004. LNCS, vol. 3298, pp. 683–697. Springer, Heidelberg (2004)
8. Giunchiglia, F., Shvaiko, P., Yatskevich, M.: S-Match: An Algorithm And An Implementation of Semantic Matching. In: Dagstuhl Seminar Proceedings of Semantic Interoperability and Integration 2005 (2005)
9. Hepp, M.: GoodRelations: An Ontology for Describing Products and Services Offers on the Web. In: Gangemi, A., Euzenat, J. (eds.) EKAW 2008. LNCS (LNAI), vol. 5268, pp. 329–346. Springer, Heidelberg (2008)
10. Horrigan, J.B.: Online Shopping. Pew Internet & American Life Project Report 36 (2008)
11. Kalfoglou, Y., Schorlemmer, M.: Ontology Mapping: The State of the Art. The Knowledge Engineering Review 18(1), 1–31 (2003)
12. Kilgarriff, A., Rosenzweig, J.: Framework and Results for English SENSEVAL. Computers and the Humanities 34(1-2), 15–48 (2000)
13. Lesk, M.: Automatic Sense Disambiguation using Machine Readable Dictionaries: How to Tell a Pine Cone from an Ice Cream Cone. In: 5th Annual International Conference on Systems Documentation (SIGDOC 1986), pp. 24–26. ACM (1986)
14. Levenshtein, V.: Binary Codes Capable of Correcting Deletions, Insertions, and Reversals 10(8), 707–710 (1966)
15. Madhavan, J., Bernstein, P.A., Rahm, E.: Generic Schema Matching with Cupid. In: 27th International Conference on Very Large Data Bases (VLDB 2001). pp. 49–58. Morgan Kaufmann Publishers Inc. (2001)
16. Melnik, S., Garcia-Molina, H., Rahm, E.: Similarity Flooding: A Versatile Graph Matching Algorithm and its Application to Schema Matching. In: 18th International Conference on Data Engineering (ICDE 2002). pp. 117–128. IEEE (2002)
17. Miller, G.A.: WordNet: A Lexical Database for English. Communications of the ACM 38(11), 39–41 (1995)
18. Niles, I., Pease, A.: Towards a Standard Upper Ontology. In: International Conference on Formal Ontology in Information Systems 2001 (FOIS 2001). ACM (2001)
19. Noy, N.F., Musen, M.A.: The PROMPT Suite: Interactive Tools for Ontology Merging and Mapping. International Journal of Human-Computer Studies 59(6), 983–1024 (2003)
20. Park, S., Kim, W.: Ontology Mapping between Heterogeneous Product Taxonomies in an Electronic Commerce Environment. International Journal of Electronic Commerce 12(2), 69–87 (2007)
21. Rahm, E., Bernstein, P.A.: A Survey of Approaches to Automatic Schema Matching. The VLDB Journal 10(4), 334–350 (2001)
22. Shvaiko, P., Euzenat, J.: A Survey of Schema-Based Matching Approaches. In: Spaccapietra, S. (ed.) Journal on Data Semantics IV. LNCS, vol. 3730, pp. 146–171. Springer, Heidelberg (2005)
23. VijayaLakshmi, B., GauthamiLatha, A., Srinivas, D.Y., Rajesh, K.: Perspectives of Semantic Web in E- Commerce. International Journal of Computer Applications 25(10), 52–56 (2011)
24. Yu, Y., Hillman, D., Setio, B., Heflin, J.: A Case Study in Integrating Multiple E-commerce Standards via Semantic Web Technology. In: Bernstein, A., Karger, D.R., Heath, T., Feigenbaum, L., Maynard, D., Motta, E., Thirunarayan, K. (eds.) ISWC 2009. LNCS, vol. 5823, pp. 909–924. Springer, Heidelberg (2009)
25. Zhang, G.Q., Zhang, G.Q., Yang, Q.F., Cheng, S.Q., Zhou, T.: Evolution of the Internet and its Cores. New Journal of Physics 10(12), 123027 (2008)