Automated Product Taxonomy Mapping in an E-commerce Environment

Steven S. Aanen, Damir Vandic, Flavius Frasincar

Erasmus University Rotterdam
PO Box 1738, NL-3000 DR
Rotterdam, The Netherlands

Abstract

Over the last few years, we have experienced a steady growth in e-commerce. This growth introduces many problems for services that want to aggregate product information and offerings. One of the problems that aggregation services face is the matching of product categories from different Web shops. This paper proposes an algorithm to perform this task automatically, making it possible to aggregate product information from multiple Web sites, in order to deploy it for search, comparison, or recommender systems applications. The algorithm uses word sense disambiguation techniques to address varying denominations between different taxonomies. Path similarity is assessed between source and candidate target categories, based on lexical relatedness and structural information. The main focus of the proposed solution is to improve the disambiguation procedure in comparison to an existing state-of-the-art approach, while coping with product taxonomy-specific characteristics, like composite categories, and re-examining lexical similarity and similarity aggregation in this context. The performance evaluation based on data from three real-world Web shops demonstrates that the proposed algorithm improves the benchmarked approach by 62% on average F_1-measure.

Keywords: products, semantic web, schema, ontology, matching, mapping, merging, e-commerce, web shop

1. Introduction

Recently, the Web has experienced a rapid growth, playing an increasingly important role in our society. The expectations are that the amount of information available on the Web will continue to grow exponentially; doubling in size roughly every five years [1]. This expresses the need to keep all this information structured. The vision of the Semantic Web from Berners-Lee et al. [2] addresses this need, with the goal to make the Web more structured, interactive, useful, and containing meaningful data, understandable for both human and computer. This has lead to the usage of ontologies [3]: standardized representations of knowledge, in which concept relationships are explicitly defined. As a result, various research in the fields of ontology management and data annotation has been performed: the matching, construction and integration of ontologies [4, 5], annotation of (Web) data [6, 7], as well as different applications of the Semantic Web [8, 9]. Unfortunately, the current Web has not yet evolved to the Semantic Web, since there is a lot of information that is not semantically annotated. Consequently, data has to be interpreted by humans, since machines do not understand them. Because machines do not understand the information embedded on Web pages, search engines are not always capable of finding the information that suits the user's needs the best.

Because machines currently do not understand true meaning of data, in the field of e-commerce, keyword-

Email addresses: `aanen@appophetweb.nl` (Steven S. Aanen), `vandic@ese.eur.nl` (Damir Vandic), `frasincar@ese.eur.nl` (Flavius Frasincar)

based search is often used. This type of search leads to a large fraction of customers that fail to find the product that fits their needs optimally [10]. One of the reasons for this is that Web-wide parametric product search is not possible. Therefore, current product comparison tools are primarily based on pricing, instead of on product characteristics. The result is that customers will be forced to compare mostly on prices, as they can not scan thousands of products themselves. Although the price competition is — economically seen — not unwanted, it can well be that customers are prepared to buy more expensive products if those would fit their needs better. Selling more expensive products will increase revenue of online retailers, and thereby contribute to the economy. Thus for both online retailers and for customers, better product search, comparison and recommendation applications on the Web are desired.

To build product search, recommendation, or comparison tools, it is needed to deal with product categorization. In general, Web sites that deal with products, such as manufacturer pages or Web stores, have a hierarchy in which products are categorized. In this way, users are able to efficiently filter the kind of products that are desired, even though possibly many thousands of products are offered. These hierarchical categorizations are called taxonomies: tree-like structures in which concepts have supertype-subtype relationships. Taxonomies are related to schemas, in which richer concept relations, with also for example cardinality constraints, lead to a graph-like structure. To be able to aggregate information from multiple Web sites dealing with products, it is needed to merge their corresponding taxonomies in order to determine to which class the collected products belong to.

In e-commerce, taxonomies are often very heterogeneous, since no standardizations are being used, and hierarchies are often manually created. In the fields of ontology and taxonomy/schema matching, many different algorithms have been proposed to deal with the heterogeneity of information structures [11, 12, 13, 14, 15]. However, since product taxonomies have some unique characteristics, such as composite categories (e.g., 'Electronics & Computers) and loose relationships (e.g., subcategory 'Hardware' under category 'PC Games', which is not a true subtype relation), specialized approaches are required.

Based on the algorithm from Park and Kim [16], which has been designed specifically for taxonomy matching in e-commerce, this paper will propose an improved approach, as there are several aspects in the Park & Kim algorithm that can be made better. More specifically, the focus of this paper will be on one of the major drawbacks of the existing algorithm: the word sense disambiguation process. The disambiguation is needed to find synonyms of the correct sense for category names. This is to account for the fact that different taxonomies make use of different words to characterize their classes, while having the same meaning. For example, 'Tools' can have completely different meaning depending on the parent category (e.g., 'gardening' vs. 'electronics'). Some Web shops might explicitly use 'Electronic Tools' or 'Gardening Tools' while others might just use 'Tools' and exploit the hierarchy to convey the intended meaning. The assumption is that when this process is improved, the overall recall and precision of the algorithm will rise as well. Apart from focusing on improving particularly this part of the algorithm, the goal is to also re-examine concepts such as composite category handling (neglected by Park & Kim), cope with depth variance between taxonomies, and propose new lexical similarity and similarity aggregation functions that better fit the e-commerce setting.

This paper is organized as following. First, we discuss in Section 2 related approaches for taxonomy/schema and ontology matching, as well as word sense disambiguation techniques and different lexical and semantic similarity measures. Similarity measures are needed to score candidate target categories for a given source category. Section 3 explains the implementation of the proposed algorithm, as well as the underlying ideas. In Section 4 the proposed algorithm will be evaluated against similar approaches using real-world data. Last, conclusions and possible future work are discussed in Section 5.

2. Related Work

Product taxonomy mapping is part of the research fields of ontology and taxonomy/schema matching. Conceptual matching in general is used in various domains of information technology, like Semantic Web applications, ontology management, e-commerce, data warehousing, and database integration. Therefore, quite a lot of research has been done on the topic in the past decades [13, 14]. The main difference between ontology and taxonomy/schema matching can be found in the semantics.

Ontologies have the meaning of concepts and relations between them explicitly encoded in their data representation. Therefore, matching algorithms can choose to primarily use knowledge from within the ontology. Ontologies are logical systems, and can be seen as a logical set of axioms according to which data is annotated, as Shvaiko and Euzenat [14] explain.

In taxonomy/schema matching however, data is often not annotated for meaning, besides the basic is-a relationships (and sometimes additional constraints as in database schemas), making it less structured than working with ontologies. Matching algorithms have to find out the meaning using external data or using the context of the data concepts within the schema. In other words, in ontology matching, computers work with data which they can understand. In schema matching, only hierarchical data is available of which a computer must first determine most relations and the meaning on its own.

Although there are some signs of initial research effort [8], in the field of e-commerce, the ideas of the Semantic Web are at their infancy in practice. Since no good applications exist at this moment, few Web stores have annotated their product pages with semantics, as specified by a product ontology like GoodRelations [17]. These semantics describe for example the relations between different products. Some exceptions do exist, but widely seen, the information on the Web — especially in product environments — is still not understood by computers.

For the above reason, taxonomy matching is more applicable than ontology matching in this field. However, both in ontology and in taxonomy/schema matching, the goal is to find relatedness between concepts, often using word sense disambiguation techniques and lexical or semantic similarity measures. Therefore, some ideas from the ontology matching domain can be applicable to taxonomy/schema matching as well. For this reason, we will discuss projects from both ontology matching and taxonomy/schema matching fields. In addition, since many of the approaches rely on relatedness of concepts and words, some measures for these will be discussed as well. As this research focuses on enhancing the word sense disambiguation process within the matching algorithm, we will also discuss some approaches for dealing with polysemy. Last, this section will give a brief overview of WordNet [18], which is a semantic lexicon used by many matching algorithms and disambiguation techniques, including the proposed solution.

2.1. Ontology Matching

This section discusses some approaches that deal with matching of ontologies. While this research focuses on taxonomy mapping, ontology alignment is a strongly related field of research which can give further insight in possible approaches of product taxonomy mapping.

As part of the *Protégé* environment for knowledge-based systems [19], *PROMPT* was developed by Noy and Musen [20]. PROMPT is a framework for multiple ontology management, including various tools to deal with tasks that often occur in management of ontologies. One of these tools, *iPROMPT*, is an interactive approach for ontology merging. It guides the user through the merging process, by making suggestions on what should be merged, and identifying problems and inconsistencies. This information is based on the structure of the concepts, and relations between them within the ontology, as well as previous user actions. iPROMPT however only looks at the local context within the ontology, which is seen as a graph, for its decisions. In other words, it only takes direct relations of concepts into account. iPROMPT is very much user-dependent, and therefore not very suitable in the domain of automated product taxonomy matching. For this purpose, often large amounts of data have to be processed within a small amount of time, which is very hard when depending on humans.

The PROMPT-suite [21] provides yet another tool for ontology merging, which is meant to deal with the deficiencies of iPROMPT: *AnchorPROMPT*. AnchorPROMPT uses a larger context within the graph for its suggestions, and is meant for larger scale comparisons. It starts with pairs of related terms as initial mappings; 'anchors', either determined by the user, or using a string or linguistics-based technique (like edit- distance tests). By analyzing the frequency of terms that occur in similar positions in the source paths, new pairs are given by the algorithm as final result. These new pairs can for example be fed back into iPROMPT to make new suggestions to the user, or they can be used as mapping themselves. Although AnchorPROMPT require less manual input than iPROMPT, it is still not a fully automatic matcher. Apart from that, AnchorPROMPT can be used for (automated) taxonomy mapping as it does not deploy the richer relations in an ontology, and it performs quite well in that respect [16]. For that reason the proposed algorithm from this paper will be evaluated against AnchorPROMPT in Section 4.

H-MATCH [22] is another ontology alignment algorithm, which uses the HELIOS [23] framework for ontology sharing and evolution in a peer-to-peer network. H-MATCH captures the relations between concepts using a linguistic and a context interpretation. For the linguistic part, it uses WordNet to capture terminological relations between terms. For the context interpretation it scores the affinity between the context of two concepts within the ontology, to measure the relatedness of the concepts themselves. Unfortunately, the context interpretation part is only usable when working with formally annotated data like in an ontology. Therefore, the algorithm can not be used for e-commerce applications which are often based on taxonomies. The linguistic part uses the same WordNet relations as the proposed algorithm in this paper: synonymy, hypernymy, meronymy, and their inverses.

OBSERVER [24] was built to deal with the semantic integration problems that occur in domain-specific ontology matching. The system uses query-expansion using two ontologies: when a user tries to query the first ontology using terms from that dataset, OBSERVER will try to expand the query results to the second ontology as well. For this purpose it uses concept term relationships such as synonyms, hypernyms, and hyponyms. The idea of using a query-based approach might be useful in certain e-commerce applications like price comparison or recommender systems. However, to reach maximum performance and reliability, which is very important for Web purposes, it seems a better idea to create a match between two taxonomies beforehand, instead of during runtime. The semantic relationships used in OBSERVER, synonymy, hypernymy and hyponymy, are also used in their proposed algorithm.

LOM [25] is a semi-automatic mapping algorithm that uses lexical similarities between concepts, and requires a 'human mapping engineer'. LOM is based on four matching procedures: whole term matching, word constituent matching, synset matching, and type matching. Whole term matching is trivial as it refers to exact string matching. Word constituent matching is somewhat more advanced and separates each word and handles it individually. Using WordNet, synonym sets are matched, which makes sure that the meaning of two terms is the same, since all words in a WordNet synonym set represent the same meaning. The last process uses the type relations that are defined within the ontology, reflected against a standardized upper ontology: SUMO [26]. This last part is not applicable in taxonomy matching, since the relational data is not available. For the whole term matching, word constituent matching, and synset matching, we can identify parts from the proposed algorithm in this paper that work towards the same goals. Apart from that, the requirement of a mapping engineer and the use of type-relational data makes LOM not useful for the e-commerce domain.

HCONE [27] is an ontology matching tool that uses WordNet together with Latent Semantic Indexing (LSI) [28] to map concepts to a WordNet representation before linking it to a target data set concept. LSI is an algorithm for automated retrieval and indexing of documents, which uses vectors of factor weights for the individual entities. HCONE uses LSI for word sense disambiguation, by replacing 'documents' with ontologies, and 'entities' with the stem of terms that form the concept labels. Furthermore, it uses descriptive logics, which can be employed for ontology matching, but not for taxonomy matching. The authors claim that fully automatic matching is impossible, hence a certain degree of human involvement will always be required.

2.2. Taxonomy/Schema Matching

Park and Kim [16] suggest an algorithm for schema matching specifically designed for product taxonomies. They start by aiming to disambiguate the name meaning of a category from the source taxonomy. This is done by comparing ancestors of that category to the hypernym hierarchy of the category's name in WordNet. The hypernym structure of which most matches with the ancestors can be found, is labeled as the correct sense. The associated synonym set is used to find candidate categories in the target taxonomy. The best of the candidates is then found using two measures: co-occurrence and order-consistency. Co-occurrence measures the similarity between the paths from source and candidate taxonomy, based on the number of co-occurring nodes. The order-consistency measures to which extent the co-occurring nodes also appear in the right order for each path.

The algorithm of Park & Kim is very applicable to the e-commerce domain, and works better than general ontology or schema matchers as reported experiments have shown [16]. However, it has some drawbacks that make it perform less good in practice. One of those is the fact that the algorithm does not handle composite categories (such as 'Electronics & Computers'), which often occur in product taxonomies, separately. The algorithm has also difficulty with mapping shorter paths, or higher level categories, due to the implementation of the word sense disambiguation process which needs more information content than is actually present for shorter paths. In addition, the algorithm prefers to map to more general categories, which is not always desirable. For example, it can map 'Garden' to 'Home, Garden & Tools' while the last category may also have a 'Garden' subcategory beneath. Besides the downsides, the algorithm of Park & Kim has shown superior results for the domain of product taxonomy matching. The proposed algorithm in this paper is based on the approach from Park & Kim, and will be evaluated against it.

COMA [35] is another approach to schema mapping, where multiple matchers for automatic or semi- automatic schema matching are employed. It has been designed for database and XML schemes. The COMA library contains approaches that use string-based techniques, but also language-based ones. The different approaches are aggregated to get a hybrid matcher with very promising results in comparison to stand-alone algorithms. Aumueller et al. [36] have improved the original version of COMA to make it perform even better, and to be compatible with ontology languages like XML Schema and OWL. However, both versions use some information that is not always available in e-commerce environments, such as element properties. This makes it less usable for our domain.

Melnik et al. propose a graph-based database schema matcher [37]. It starts with an initial mapping provided by a simple string matcher. The mappings are then refined by checking the similarity between adjacent elements of the entities in the schema: directly related concepts. Since this can deliver multiple possible mappings, a threshold is computed to pick the ones most likely to be correct, and filter the rest out. Unlike many other approaches, SimilarityFlooding does not take terminological variations into account. Since this is often desired in e-commerce, due to the varying terminology, and since SimilarityFlooding is more applicable to database schemes because of the graph-like approach, this algorithm is not very useful for mapping product taxonomies, which only have a hierarchical structure. The idea of using similarity between adjacent concepts, like the similarity between a parent of a source class and a parent of a candidate target class, may however be applied. However, that would also make an algorithm slower, which is generally undesired in e-commerce domain.

There are some schema matching approaches that rely on more data then may be available for product taxonomies, such as class properties. For example, a category for films might have attributes like actors, which other categories do not necessarily have. While it is a valid assumption to use this data, it is not always available in practice. Cupid [38], a hybrid matcher, uses linguistic measures but also takes cardinality relations from database schemes and data-types into account. S-Match [39] also uses parameters of classes and their data types, next to the use of disambiguation techniques. The focus, however, is on the last one: semantic variations between concept terms.

Solutions that take a totally different approach have been proposed by Li and Clifton [40], mainly focusing on neural networks. DCM [41] uses correlation mining and co-occurrence patterns for matching. CTX- Match [42] combines a lot of linguistic approaches such as sense disambiguation, part-of-speech tagging, and multi-word recognition. Other work and ideas on schema matching can be found in surveys such as by Rahm and Bernstein [11], which discuss different applications and approaches for schema matching and categorize different algorithms according to the used techniques. Shvaiko and Euzenat [14] explain the differences between ontology and schema matching, and the underlying principles of matching these structures. Some different techniques are discussed, and different algorithms are summarized. Do et al. [13] evaluate approaches meant for different application domains, using different data sets according to those domains, measuring their recall, precision and F_1-measure. Noy [12] explains different approaches that can be used for taxonomy/schema matching. Avesani et al. [43] propose a dataset for the evaluation of schema matchers, and experiment with it using three different algorithms.

To summarize, there are no previous studies related to our work that cover all aspects that are import in e-commerce taxonomy mapping: (1) using only the categories in the taxonomy as input, (2) scalability and efficiency (i.e., algorithms must be able to handle thousands of categories with no exponential growth in computational time), and (3) account for inconsistencies in the taxonomy (e.g., categories that appear on multiple paths in the taxonomy). With our approach we aim to handle all three cases. Furthermore, because the primary focus for improvement in this paper is the word sense disambiguation process, the insights gained from this study will contribute to the theoretical understanding of word sense disambiguation both in general sense, as well as specifically applied to e-commerce taxonomies.

2.3. Similarity Measures

As became clear from the previous sections, many matching algorithms make use of one or more measures of lexical or semantic similarity. There are many of these measures available, but we will only highlight some of them which seem to be most usable in the domain of this paper. The Jaccard index [44] is a statistic that measures the similarity between two sets. The index (or coefficient) is given by:

$$\text{jaccard}(A, B) = \frac{|A \cap B|}{|A \cup B|} \tag{1}$$

This produces an index in the range $[0, 1]$, where 1 would mean that the two sets are identical. The Jaccard coefficient can be used for lexical string matching by employing each individual word as an element from a string A or B. This is one way in which the proposed algorithm in this paper uses Jaccard. Another way, which is also being used in the algorithm, is by using each individual character from both strings as elements. In both cases, however, Jaccard only measures the co-occurrence of the elements, while ignoring any order information. However, in practice, Jaccard is useful in the taxonomy mapping domain since it provides a good measure of relatedness with an easy and fast computation.

The Dice [45] coefficient is another similarity measure, related to the Jaccard index. It is also based on set theory. The coefficient is given by:

$$\text{dice}(A, B) = \frac{2\,|A \cap B|}{|A| + |B|} \tag{2}$$

When used for strings, bi-grams are often used as elements: groups of two characters or words. It is also possible to use individual characters or words as elements.

The Levenshtein distance [46] is a metric that expresses the amount of difference between two strings. It is defined as the minimum number of edit operations needed to transform one string into the other (often called the edit distance). The operations that are possible are: insert, delete or substitute an individual character. For example, the Levenshtein distance between 'lamp' and 'ramp' is 1, since the only edit that is needed is to substitute the first character. The distance metric theoretically does not have a limit, since individual strings can be infinitely long, which is why a normalized version of the Levenshtein distance is often used. When the absolute distance is given by $\text{lev}_{\text{abs}}(A, B)$, the normalized distance is given by

$$\text{lev}_{\text{norm}}(A, B) = \frac{\text{lev}_{\text{abs}}(|A|, |B|)}{\max(|A|, |B|)} \tag{3}$$

which corrects the metric for length of the largest string, and makes it an index with a value between 0 and 1. Just as with Jaccard and Dice, the Levenshtein distance can also be used with whole words as elements instead of characters. Implementation of Levenshtein is however a bit more difficult and its computation is slower than Jaccard.

Resnik [47] proposes a measure of semantic relatedness based on the amount of 'information content'. The information content is a value that is assigned to each concept in a hierarchy based on evidence that can be found in a corpus. The idea is that concepts with a high information content are very domain-specific, while a low information content would indicate a very general concept. By counting the frequency at which a concept occurs in a corpus, the probability p of encountering this concept is estimated. When sense-tagged data is used, the frequency count is quite simple. The information content of a concept c is given by:

$$\text{IC}(c) = -\log p(c) \tag{4}$$

The Resknik measure can be applied using WordNet as corpus. The similarity between two concepts, is then given by the information content of the lowest common subsumer of both concepts; the lowest concept in WordNet that subsumes both concepts. The function is given by:

$$\text{resnik}(c1, c2) = IC(lcs(c1, c2)) \tag{5}$$

where `lcs()` gives the lowest common subsumer of concepts $c1$ and $c2$. While the Resnik measure works quite well when used within a word sense disambiguation algorithm [48], the computation is quite time-consuming, making it less desirable for the e-commerce domain.

For the Resnik measure some variants have also been developed. Jiang and Conrath [49] augment the information content measure with a measure for path length between concepts. The function is given by:

$$\text{jc}(c1, c2) = IC(c1) + IC(c2) - 2 \cdot IC(lcs(c1, c2)) \quad (6)$$

which uses the same functions as described previously.

Lin [50] doubles the information content of the lowest common subsumer and divides that by the information content of both concepts themselves. This is known as the Dice-coefficient. The formula is given by:

$$\text{lin}(c1, c2) = \frac{2 \cdot IC(lcs(c1, c2))}{IC(c1) + IC(c2)} \quad (7)$$

A more simple approach for semantic relatedness is that from Leacock and Chodorow [51], which uses the normalized path lengths of noun concepts in a hierarchy like that of WordNet. The more related two concepts are, the lower the number of nodes that have to be traversed to go from one to the other. The function is given by:

$$\text{lc}(c1, c2) = -\ln\left(\frac{minPathLength(c1, c2)}{2 \cdot d}\right) \quad (8)$$

where `minPathLength()` computes the shortest path between two synonym sets $c1$ and $c2$, and d is the depth of the tree.

These and other measures for semantic relatedness are evaluated for usage in an adapted version of Lesk for word sense disambiguation [48]. Lin [52] discusses what 'similarity' actually means in different contexts, and explains some approaches that can be used in a specific context.

2.4. Word Sense Disambiguation

Word Sense Disambiguation (WSD) has been applied in numerous other studies (e.g., news classification [53], sentiment analysis [54], etc.) but not in taxonomy mapping. Most approaches are meant to disambiguate individual words in a text. For product taxonomy mapping, there is usually not a plain piece of text available to use. Instead, the context consists of for example ancestors and children of category nodes. In addition, speed and recall performance is very important in e-commerce. Recall is very important, as it is important to include as many products as possible. Precision on the other hand, whether the products are classified correctly, is of less importance. Speed is a relevant factor, since querying on the Web must be as fast as possible. Since many disambiguation algorithms do not qualify for these conditions, only a few interesting approaches will be discussed in this section.

Lesk [55] proposed an algorithm that disambiguates a term based on the overlap of words from its gloss with glosses of surrounding words. The idea is that words that appear in one sentence should be semantically related to each other. A target word is assigned the sense with the most gloss word overlaps with the context. The original Lesk algorithm uses glosses found in traditional dictionaries. This can however be a limitation of the method since glosses are often very compact and may not include sufficient vocabulary to find overlaps.

Banerjee and Pedersen [56] adapted the original Lesk algorithm to work with the rich semantic data relations that WordNet offers. This approach uses a general 'window' of words which is used as the context, instead of words from a sentence. This window also contains the target word to disambiguate. Each of the words from the window can be present in one or more possible synonym sets, which reflect a sense in WordNet. Each of these words can thus have multiple glosses associated as well. A combination of synonym sets (a synset has one sense of a word in WordNet) is seen as a candidate combination, and will be scored. The total number of candidate combinations is equal to the product of the number of senses for each word from the window: $\prod_{i=1}^{n} |W_i|$ where W_i is a word from the window and $|W_i|$ is the number of possible senses for that word. For every candidate combination (so every combination of senses for each word in the window), the gloss overlaps are computed between every word in the window. This is not only done for glosses of the synonym sets themselves, but also for related synonym sets such as hyponyms, hypernyms, meronyms, and holonyms. The overlap between glosses is measured as the squared number of words in the longest sequence of consecutive words that appear in both glosses. While this approach works well, it is clear that the enormous amount of combinations and therefore gloss overlaps that have to be computed puts a lot of pressure on the performance.

In an evaluation of similarity measures in combination with the adapted Lesk algorithm just discussed [48] the authors explain the computation for gloss overlap as 'just another similarity measure'. They state that other measures can be used as well. This is actually what the proposed algorithm in this paper does. It uses the WordNet-adapted version of Lesk, with Jaccard as similarity measure, because of its speed performance. While other measures have been tried as well during our research, the enormous amount of needed computations made it necessary to use an extremely fast computation, and on top of that, a very small window of context words.

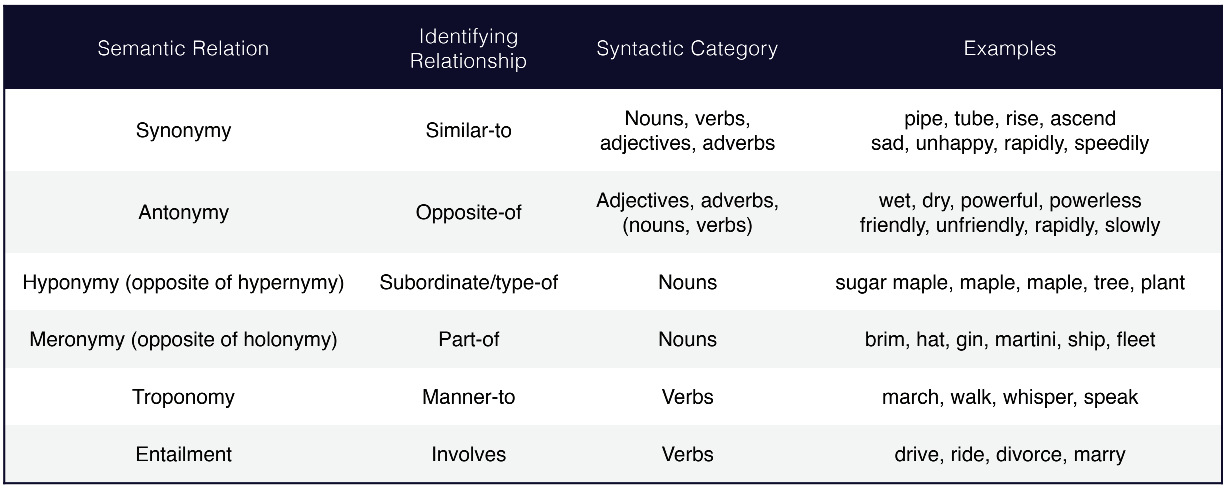
Simplified Lesk [57] uses heuristics to make Lesk faster and simpler. The different meanings of context words, or their glosses, are not taken into account, and are matched purely on exact lexical base to the glosses of the target word. The approach works good, even in comparison to original Lesk [58], but is a lot faster. Since it does not use richer semantic relations, such as the approach from Banerjee and Pedersen [56], it is seen as less useful for the proposed algorithm in this paper.

While Lesk is one of the most recognized algorithms for word sense disambiguation, many others exist. One of these algorithms uses the conceptual distance in the WordNet graph [59]. Yarowsky [60] uses statistical models and weights to classify words to the Roget thesaurus' categories. It uses a context of 100 words which would be very large for product taxonomy purposes. Leacock et al. [61] use a statistical classifier to combine cues like related words from the context with terms from the concept to disambiguate a term using WordNet. Navigli and Velardi [62] developed an approach using pattern recognition and digraphs. [63] presents a survey on word sense disambiguation which describes many different approaches and applications of disambiguation. For the evaluation of word sense disambiguation approaches, sense-tagged corpora exist [64], and even a conference is organized on a regular base [65].

2.5. WordNet

WordNet [18, 66] is a semantic lexicon; it is basically a dictionary, in which relations between words are known. The lexical database contains English nouns, verbs, adjectives, and adverbs, organized into synonym sets (or synsets); sets of words sharing the same meaning. WordNet contains over 118,000 words forms in about 90,000 synsets. Between these synsets, the following semantic relations are described: synonymy (symmetric, word with same meaning), antonymy (opposing-name), hyponymy (sub-name) and hypernymy (super-name), meronymy (part-name) and holonymy (whole-name), troponymy (manner-name) and entailment relations between verbs. Examples of these relations are given in Figure 1. WordNet is used by many of the ontology and schema matching, and word sense disambiguation algorithms. The proposed matcher in this paper also uses WordNet for sense disambiguation, collection of synonyms and word morphological processing into lemmas.

WordNet versions for different, sometimes multiple languages have also been made, such as MultiWordNet [67], EuroWordNet [68], the BalkaNet [69] project and FinnWordNet. Universal WordNet (UWN) [70] is an automatically constructed WordNet variant covering many different languages with over a million words in total. Next to this, different projects have linked WordNet to other datasets, such as ontologies. This has also been done for the upper ontologies of SUMO [26] and DOLCE [33] which have been previously mentioned.



Semantic Relation	Identifying Relationship	Syntactic Category	Examples
Synonymy	Similar-to	Nouns, verbs, adjectives, adverbs	pipe, tube, rise, ascend sad, unhappy, rapidly, speedily
Antonymy	Opposite-of	Adjectives, adverbs, (nouns, verbs)	wet, dry, powerful, powerless friendly, unfriendly, rapidly, slowly
Hyponymy (opposite of hypernymy)	Subordinate/type-of	Nouns	sugar maple, maple, maple, tree, plant
Meronymy (opposite of holonymy)	Part-of	Nouns	brim, hat, gin, martini, ship, fleet
Troponomy	Manner-to	Verbs	march, walk, whisper, speak
Entailment	Involves	Verbs	drive, ride, divorce, marry

Figure 1: Examples of semantic relations in WordNet as presented by Miller [66]

3. Framework

In this section, we discuss the proposed algorithm for taxonomy mapping. As mentioned earlier, it is based on the approach from Park and Kim [16]. The reason why we took this approach as a base is that this algorithm has been specifically designed for matching product taxonomies and works very good for that domain in comparison to a general schema or ontology matcher. In addition, the concepts that the Park & Kim algorithm uses are both quite responsive and well-performing, combining word sense disambiguation with path similarity and lexical similarity to find the best possible mapping from one taxonomy path to the other. The reason why path similarity works well when combined with lexical similarity is that they complement each other. Lexical similarity addresses the issue that categories are represented differently across Web shops. Path similarity, on the other hand, takes a more high- level approach and analyzes which of the candidates is the best fit with respect to the context (surrounding categories in the taxonomy). The section is organized in the order of the algorithm itself, as can be seen in Figure 2. First, the category name from the source taxonomy is disambiguated so synonyms of the correct sense can be used for the next step: candidate selection. A set of candidate paths in the target taxonomy is selected based on the synonyms just found. Last, the candidates are scored based on the *co-occurrence* and *order-consistency* metrics with respect to the source taxonomy path. These measures are combined to select the best mapping. The procedure as shown in the figure is executed for every path from the source taxonomy. First however, we will explain the general assumptions that were used for the development of the proposed (improved) algorithm.

3.1. General Assumptions

Composite Categories. One phenomenon that can be found regularly in product taxonomies is that of composite categories. Composite categories are multiple — usually related — classes of products that are categorized as one. An example: the 'Movies, Music & Games' category from the product taxonomy of Amazon.com [71]. Each of the three parts could have been a separate category, and therefore the assumption is that they should be treated like that in the algorithm. The problem with the Park & Kim algorithm, which neglects this phenomenon, is that it is not able to disambiguate composite category names as a whole. Furthermore, the target taxonomy may not use the same conjunction of classes, which can make

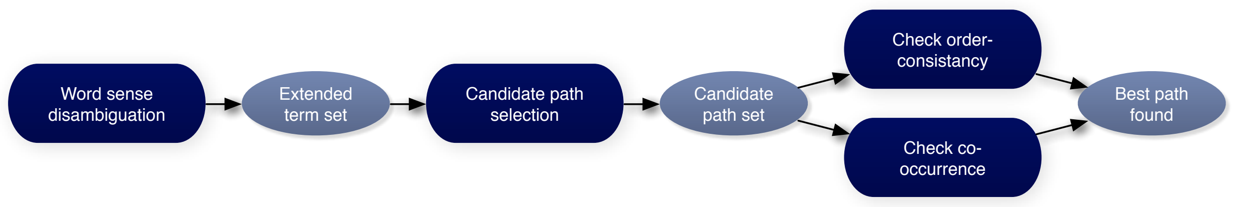


Figure 2: Framework overview of the proposed algorithm in this paper and the algorithm from Park and Kim [16]

it hard to find candidate categories. Therefore, the proposed algorithm in this paper splits the composite category names on ampersands, commas, and the word 'and'. The result, the set of classes that makes the composite category, is referred to as the *split term set*.

Root Categories. Product category hierarchies use tree-structured relations, and should therefore always have a root category, although that is not always visible on Web sites. The root categories, however, have no meaning with respect to the products that fall beneath them. Examples of root categories are 'Products', 'Shopping', and 'Online Shopping', as in the datasets that were used in this research. Since the root categories have no meaning, they are not used in the matching processes in our proposed algorithm, in contrast to the approach from Park & Kim. Furthermore, in the proposed algorithm, the root from the source taxonomy, which is on itself a category and should therefore be handled, is always mapped to the root from the target taxonomy. As an example, Figure 3 makes clear that the root categories on the top in dark blue (dark gray in black & white printing) are matched, even though the category names are not matching syntactically.

Parent Mapping. Product taxonomies differ also in terms of the category hierarchy depth. Some might have split products into very specific sub-categories while other taxonomies have a more general classification. Whenever it occurs that a specific category exists in the source taxonomy, but not in the target taxonomy, the algorithm proposed in this paper assumes that the source category should be mapped to a more general category from the target taxonomy. As an example, Figure 3 shows that the category 'Books' from Overstock is mapped to 'Books' from Amazon, as it should. 'Humor Books' is however a more specific category which Amazon does not have. But in the end, a humor book is still a book, so it is assumed that all humor books from Overstock fit into the more general 'Books' category from Amazon. This assumption is not used by Park & Kim. The more general category is found by looking at the target category to which the parent of a source category is mapped to.

Case Sensitivity. A general assumption for the proposed algorithm is that the usage of capitals in category names does not affect the meaning. Therefore, all matching within the algorithm ignores case sensitivity. This also solves the problem that many category names in taxonomies start with a capital, while WordNet uses lowercase characters for words. The algorithm of Park & Kim does not address this issue.

3.2. Word Sense Disambiguation

The first step in the mapping algorithm is to disambiguate the name of the category that is currently being processed, which is necessary in order to obtain the synonyms from the correct sense of the category. This improves the ability to match categories that are synonyms of each other. For example, when searching for 'notebooks' in the sense of a portable computer, it is desired that also 'laptops' will be found. However, when all synonyms of the word 'notebook' would be used, synonyms with a different meaning; like that of a book to write notes in, would

Table 1: Ratio of senses found per source taxonomy for each algorithm

Data set	Park & Kim	Aanen–Park	Aanen–Lesk
Amazon	0.048	0.026	0.430
ODP	0.104	0.099	0.563
O.co	0.020	0.018	0.241
Average	0.057	0.048	0.411

also be included in the search results. For acquiring the synonym sets, we use WordNet [66].

Park & Kim suggest a word sense disambiguation procedure based on matching the names of ancestors of the current category to the hypernym hierarchies of the current category name. This procedure is relatively fast, but not very good, as is shown in the evaluation section (in Table 1). The method fails very often to find any match, and therefore can often not assign a sense to the category name. The problem is that higher level categories (shorter paths) have less information content that can be used to match to the hypernym hierarchies.

To illustrate the effect of using different techniques for word sense disambiguation, this section will describe two different approaches. Both approaches can be used with our proposed algorithm. In the evaluation we also consider how our algorithm performs with these two approaches for

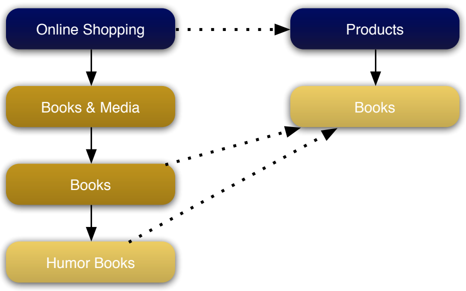


Figure 3: Mapping example from Overstock (left) to Amazon (right) categories. Normal lines indicate a parent- child relationship; dotted lines indicate a match between source and target category made by the algorithm.

word sense disambiguation. The first approach is that of Park & Kim, adapted according to the first general assumption from Section 3.1: the need to split composite categories.

The second approach is a new process, partly based on the adapted algorithm from Lesk [55], by Banerjee and Pedersen [56]. Note that in either versions, including the original process described by Park & Kim, the target taxonomy does not play a role in the word sense disambiguation procedures. The goal is to disambiguate the category name from the source taxonomy. Section 3.1 states that classes in composite categories should be handled separately. Therefore, each category that is disambiguated, is first split into the composing classes. For each of the individual classes that are acquired, the word sense disambiguation processes described in this section are executed accordingly.

3.2.1. Adapted Park & Kim Disambiguation

This section explains the word sense disambiguation procedure based on the algorithm of Park & Kim. To disambiguate a (split) category name, it is first needed to know all possible senses. Using WordNet, the different meanings are gathered in the form of hypernym structures. For each possible sense, there is a hierarchy of hypernyms, like in Figure 5, which can be retrieved from WordNet, referred to as the *hypernym hierarchy*. In the figure, two possible senses for 'piano' are shown, each belonging to a synonym set which is shown at the bottom of its hypernym hierarchy in light blue (light gray in black & white printing).

In order to find the best sense, the ancestors of the current category (referred to as *upper categories*) are matched to the hypernym hierarchies. An example of upper categories can be found in Figure 4 as the gold-colored (gray in black & white printing) nodes in the middle: from 'Entertainment' to 'Keyboard'. The function that measures the overlap between an upper category and one hypernym

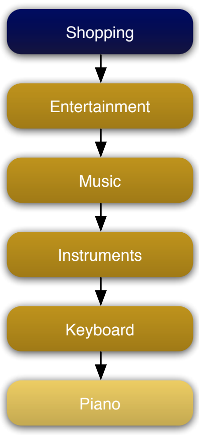


Figure 4: Category path of 'Piano' in the ODP product taxonomy

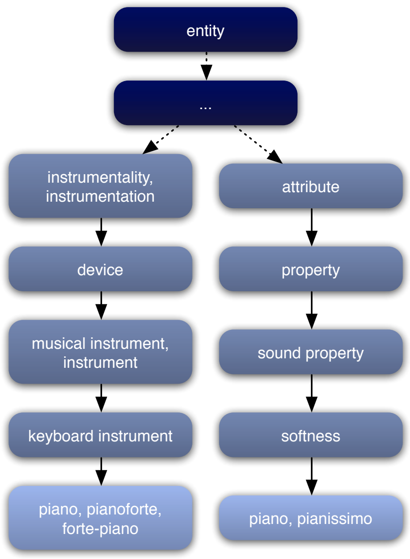

entity
...
instrumentality, instrumentation
device
musical instrument, instrument
keyboard instrument
piano, pianoforte, forte-piano
attribute
property
sound property
softness
piano, pianissimo

Figure 5: Two hypernym hierarchies for 'piano' in WordNet

hierarchy is given by:

$$\text{match}(t, S) = \{x|x \in H, H \in S \text{ and } \text{base}(t) \in H\} \tag{9}$$

where t is an upper category to match, H is the set of synonyms of one hypernym, S is a hypernym hierarchy (representing one WordNet sense), and `base()` is a function which looks up the lemma of a term using WordNet. The result of `matches()` is a set of matching nodes between an upper category and a hypernym hierarchy. In the example of Figures 4 and 5, `matches()` would return one match for upper category 'Instruments' and the left hypernym hierarchy which contains 'instrument'.

The result of Equation (9) can be used to define a metric that measures the amount of similarity between an upper category and a hypernym hierarchy. In hypernym hierarchies like those in Figure 5, nodes closer to the base term (leaf) in light blue (light gray in black & white printing) are more specific, and have more information content for that reason. Nodes closer to the root in dark blue (dark gray in black & white printing) however, are very general and do not have a lot of meaning. Therefore, nodes closer to the leaf are seen as most important. This idea is used in the function that computes the similarity between an upper category and a hypernym hierarchy given by:

$$\text{hyperProximity}(t, S) = \begin{cases} \frac{1}{\min\limits_{x \in C}(\text{dist}(x,b))} & \text{if } C \neq \emptyset \\ 0 & \text{if } C = \emptyset \end{cases} \tag{10}$$

where t is an upper category to match, S is a hypernym hierarchy (representing one WordNet sense), C is the result of match(t, S), b is the base term (leaf) of hypernym hierarchy S, and where function `min()` returns the minimum of a set of values, and `dist()` computes the distances between two nodes in the hypernym hierarchy. The distance is defined as the number of arcs that are being traversed when navigating from node to node.

For upper category 'Instruments' and the left hypernym hierarchy in Figures 4 and 5 respectively, there is only one match that can be found by Equation (9): 'instrument'. Therefore, the distance between that node and the base, 2, results in a hyper-proximity of $\frac{1}{2}$.

Using Equations (9) and (10), the similarity between one ancestor of the current category and one sense (hypernym hierarchy) can be measured. However, to measure the final score per sense (denoted by S), it is needed to measure the similarity with all the ancestors (upper categories) from the category path (denoted by P). This similarity is given by:

$$\text{pathProximity}(P, S) = \sum_{\substack{x \in P, \\ x \neq S_b}} \frac{\text{hyperProximity}(x, S)}{|P| - 1} \tag{11}$$

where P is the list of nodes from the source category path (without the root), S is a hypernym hierarchy (representing one WordNet sense), and S_b is the base term (leaf) of S. The path-proximity is the average hyper-proximity over all upper categories for one hypernym hierarchy (representing a sense). Park & Kim divide by the total number of nodes instead of the ancestor count, including root (which they do use in the source path) and base term. However, this is a minor mistake since that would not lead to an average, which is why it is changed in our algorithm.

In the example of the left sense in Figure 5, there is only one matching upper category: 'Instrument'. The hyper-proximity $\left(\frac{1}{2}\right)$ is divided by the total number of ancestors: 4, resulting in a path-proximity of 0.125. The denominator is 4 because the root node is not used, as explained in Section 3.1. The other hypernym hierarchy on the right in the figure has a path-proximity of 0, since there are no overlaps between the hierarchy and the upper categories.

Selecting the Proper Sense. The function in Equation (11) measures the fit of the source category path to one possible sense of the category term. Therefore, it is used to score all possible senses, so all hypernym hierarchies that were found earlier. The sense with the highest score is then marked as correct. In the example of Figure 5 this would be the left sense: 'piano' as a musical instrument.

From the best hypernym hierarchy that is found, only the synonym set of the leaf is used: the light blue (light gray in black & white printing) node at the bottom in the figure of the example. This is from now on being referred to as the *extended term set*. More specifically, when dealing with a composite category, the disambiguation process would be used multiple times, resulting in multiple extended term sets. For example, disambiguating category 'Movies, Music & Games' would result in three extended term sets: one for each of the classes. These synonym sets must be stored separately, so for each part of the composite category name, there is an individual synonym set.

It is possible that a (split) category name is unknown to WordNet, or that no best sense can be found. In that case, the original term is used as the 'extended term set'. According to Park & Kim, in such a scenario the whole procedure should be canceled from this point, resulting in no mapping for the current category. However, while evaluating that approach for this research, it appeared that that would yield an extremely poor recall. That is why the algorithm of Park & Kim is evaluated in the same way for this paper, i.e., by using the term itself when the disambiguation fails (for a fair comparison).

3.2.2. Adapted Lesk Disambiguation

This section explains the word sense disambiguation procedure partly based on the adapted version of the algorithm from Lesk [55], by Banerjee and Pedersen [56]. This approach is not used in the algorithm from Park and Kim [16]. The idea is quite simple: to each sense of a word, a gloss is associated providing a brief notation of the meaning. By measuring the overlap between the glossary of the category name and those of related categories, it should be possible to find the best sense. Parents and children of category nodes are usually strongly related with each other. Therefore, the assumption is that the glosses of the senses that represent those relations, have overlaps. However, since glosses are often very compact, they might not provide sufficient vocabulary to measure the overlap. Therefore Banerjee and Pedersen [56] suggest to use the richer semantic relations provided by WordNet. This increases the number of glosses that have to be compared, but it also enhances the chance of finding enough overlaps to assign the right sense to a category name. The same authors [48] also suggest that different measures of similarity can be used, instead of the strict lexical matching of glosses that Lesk uses. Because of the good accuracy and speed, this paper uses the Jaccard measure [44] to score the overlap between glosses, with the words as elements. Just as with the word sense disambiguation process described in Section 3.2.1, the procedure explained in this section disambiguates every part of a composite category, if that is the case. The word sense disambiguation process is completely described in Algorithm 1. This procedure applies the approach of Banerjee and Pedersen [56], described in Algorithm 2, which has been customized for the domain of this paper.

As becomes clear in Algorithm 1, only two words are being used as context: one part of a parent category and

Algorithm 1 Gather Extended Term Set for a Category

```
Require: category to disambiguate: $w_{\text{category}}$
Require: category parent: $w_{\text{parent}}$
Require: set of category children: $W_{\text{child}}$
Require: function splitComposite($w$) which splits composite category names into individual classes based on ampersands, commas, and conjunctions
Require: function disambiguate($w_{\text{target}}, W_{\text{context}}$) disambiguates a word using a set of context words, as described by Algorithm 2
 1: $S_{\text{category}} \leftarrow \text{splitComposite}(w_{\text{category}})$
 2: $S_{\text{parent}} \leftarrow \text{splitComposite}(w_{\text{parent}})$
 3: $S_{\text{child}} \leftarrow \emptyset$
 4: for all $w_{\text{current}} \in W_{\text{child}}$ do
 5:   $S_{\text{child}} \leftarrow S_{\text{child}} \cup \text{splitComposite}(w_{\text{current}})$
 6: end for
 7: $extendedTermSet \leftarrow \emptyset$
 8: for all $w_{\text{split}} \in S_{\text{category}}$ do
 9:   $found \leftarrow$ false
10:   {Pick a combination of one (split part of a) parent and child to use as context}
11:   for all $(w_p, w_c) \in S_{\text{parent}} \times S_{\text{child}}$ do
12:     if $found =$ false then
13:       $synset \leftarrow \text{disambiguate}(w_{\text{split}}, \{w_p, w_c\})$
14:       if $synset \neq null$ then
15:         $extendedTermSet \leftarrow extendedTermSet \cup \{synset\}$
16:         $found \leftarrow$ true
17:       end if
18:     end if
19:   end for
20:   if $found =$ false then
21:     $extendedTermSet \leftarrow extendedTermSet \cup \{w_{\text{split}}\}$
22:   end if
23: end for
24: return $extendedTermSet$
```

Algorithm 2 Target Word Disambiguation using Context

```
Require: target word to disambiguate: $w_{\text{target}}$
Require: set of context words to use: $W_{\text{context}}$
Require: synsets($w$) gives all synonym sets (representing a sense) for word $w$
Require: related($s$) gives (directly) related synonym sets of synset $s$ (based on hypernymy, hyponymy, meronymy and holonymy)
Require: gJaccard($s_a, s_b$) computes the Jaccard similarity between the glosses of two synsets ($s_a, s_b$)
 1: {For each word $W_i \in W$, fill window $W$ with synsets of the target word and its context words.}
 2: $W \leftarrow \{\text{synsets}(w_{\text{target}})\}$
 3: for all $w_{\text{current}} \in W_{\text{context}}$ do
 4:   $W \leftarrow W \cup \{\text{synsets}(w_{\text{current}})\}$
 5: end for
 6: $bestScore \leftarrow 0$
 7: $bestTargetSynset \leftarrow null$
 8: {Each candidate $C$ is a combination of choosing one synset for each word $W_i \in W$ (i.e., a $n$-ary Cartesian product of all synsets from $W$)}
 9: for all $C \in \prod W_i$ do
10:   $cScore \leftarrow 0$
11:   {Iterate over all unique pairs from $C$}
12:   for all $(s_a, s_b) \in C^2$ do
13:     $R_a \leftarrow \text{related}(\text{s}_\text{a})$, $R_b \leftarrow \text{related}(\text{s}_\text{b})$
14:     for all $(r_a, r_b) \in R_a \times R_b$ do
15:       $cScore \leftarrow cScore + \text{gJaccard}(r_a, r_b)$
16:     end for
17:   end for
18:   if $cScore > bestScore$ then
19:     $bestScore \leftarrow cScore$
20:     $bestTargetSynset \leftarrow w_{\text{target}}$ {current synset for $w_{\text{target}}$ is selected from $C$}
21:   end if
22: end for
23: return $bestTargetSynset$
```

one part of a child category. When no parent is available, namely when dealing with top categories, two child parts are used instead. The reason not to use more context words is the larger computation time; adding an extra word to the context increases the computation time exponentially. This is due to the enormous amount of combinations that have to be evaluated. However, the procedure also makes clear that when no synonym set (sense) is found using two specific context words, two other context words are tried. When none of the combinations of context words result in a successful disambiguation, only the (split part of the) category name is used as 'extended term'. One final remark must be made: theoretically it could happen that even with only two context words the computation time would be very large, for example when many synonym sets are found. To prevent the total computation time of the mapping procedure from exploding, one individual extended term set procedure is aborted when taking longer than 2000 ms to compute. Although we did not encounter this situation in our evaluation, we did implement it to make sure that the processing time is bounded.

3.3. Candidate Target Category Selection

When the used word sense disambiguation procedure has finished, an extended term is produced. The extended term set contains synonyms of the correct sense for all class parts within one category, or at least the original class names. Using this data, the target taxonomy can be searched for categories that possibly match the source category. We refer to such categories in the target taxonomy as *candidate categories.* The procedure for the candidate selection is not complicated. Basically, the complete target taxonomy is scanned for categories that match the extended term set in the following way: for each split term from the source category that is contained in the extended term set, it is checked whether one of its synonyms is part of the name (a substring) of a category from the target taxonomy.

When at least half of the number of split terms has a match, the target taxonomy's category is marked as a candidate. For example, for source category 'Films', target category 'Movies, Music & Games' would be a match, since one of the synonyms of 'Films', i.e., 'movies', is part of the target category. However, if it would be the other way around ('Movies, Music & Games' as source category), there would be no match, since only a third of the words are included in the target category.

Ideally, the classes from the source category would be a subset of those of the target category. However, in practice, that would be a too strict procedure to use. Sometimes matches can not be found, due to the used word sense disambiguation process which can not be perfect. Though according to a human, there maybe would be a match. Therefore, it is better to scan more loose (at least half a subset), and let the co-occurrence and order-consistency measures filter out the faulty candidates.

The Park & Kim algorithm uses a different method to find candidate categories, as they ignore the phenomenon of composite categories. Their procedure is simply to check whether the source category term, or a synonym from the extended term set, is contained in a target category name. Furthermore, the Park & Kim algorithm operates under the assumption that more general target categories are better than more specific ones. Therefore, only the highest categories from the taxonomy are used as candidates, and more specific ones are deleted. However, this is a weak assumption. For example: source category 'Games' would fit better to target category 'Movies, Music & Games → Games' (more specific, child) than to 'Movies, Music & Games' (more general, parent). The procedure of Park & Kim however would choose the last option: map to parent 'Movies, Music & Games'. The proposed algorithm in this paper gives every candidate category the same chance of winning.

When the candidate categories are collected, the best one has to be chosen. This is done by scoring them according to two measures, described in the next sections. The extended term set, the result from the disambiguation procedure, is only used for candidate selection. It will not be used from this point on in the algorithm.

3.4. Co-occurrence Measure

One possibility to evaluate how well a candidate target category fits to the source category, is to measure their path similarity. Co-occurrence is a measure that grades the overlap between paths. The order in which the nodes occur is however not taken into account.

The procedure to compute the co-occurrence measure starts with a function called `maxSim()`, which computes the similarity between a category name, and another category's path. Both can be from either source or target taxonomy. First, the base form (lemma) of the category name is retrieved using WordNet. Then, the character-based Jaccard similarities are computed for the lemma and each node from the other category's path. From these similarity indexes, `maxSim()` returns the highest one: the best match between the category name and a node from the other category's path. In other words, it measures how well a given category would fit into the path of another category. The function is given by:

$$\text{maxSim}(t, P) = \max_{w \in P}(\text{jaccard}(\text{base}(t), w)) \qquad (12)$$

where t is a a category name, P is the list of nodes from a taxonomy path, w is one category name from the the taxonomy path, and `base()` gives the lemma of a word, and `jaccard()` gives the character- based Jaccard index of two terms, as described by Equation (1).

Instead of the Jaccard index, the Park & Kim algorithm employs exact lexical matching. Their function returns the value of the length of the larger string divided by the length of the smaller string, if one of either is contained in the other. Since no matches would be found when words occur in a different form, for example with 'clothing' and 'clothes', exact lexical matching is seen as an inferior method in this research.

Using `maxSim()` the co-occurrence measure can be computed. It measures the level of similarity between two category paths. This is used in the proposed algorithm in this paper to measure the overlap between the source taxonomy path and a candidate target path. The function is given by:

$$\text{coOcc}(P_{\text{src}}, P_{\text{targ}}) = \left(\sum_{t \in P_{\text{targ}}} \frac{\text{maxSim}(t, P_{\text{src}})}{|P_{\text{targ}}|} \right) \cdot \left(\sum_{t \in P_{\text{src}}} \frac{\text{maxSim}(t, P_{\text{targ}})}{|P_{\text{src}}|} \right) \qquad (13)$$

where P_{src} is the list of nodes from the source taxonomy path, P_{targ} is the list of nodes from a candidate target taxonomy path, and t is a category name. This function measures both the amount of nodes from the source category that occur in the candidate category, and the other way around.

3.5. Order-Consistency Measure

Using the co-occurrence measure described in Section 3.4, the similarity of the source category path and a candidate target path can be computed. Unfortunately this is not sufficient, since the order of the nodes as they appear in a path is also important for distinguishing the meaning. Therefore, the order- consistency measure is introduced in this section. The order-consistency is a ratio of the degree in which co- occurring nodes from two paths occur in the same order.

To compute the order-consistency, it is first needed to know which nodes exist in both paths. This is computed by the function `common()`. It scans all nodes from the candidate path, and retrieves their synonyms from WordNet. When a candidate path node name, or one of its synonyms, is contained in the node name of the source category path, or the other way around, there is a match. `common()` returns a set of all matching pairs.

The next function, `precedenceRelations()`, creates pairs of nodes that were found by `common()`. Of each pair, the first node always hierarchically comes before the second node in the source path. In this way, the order of all common nodes becomes clear though binary node associations. The order in which the binary pairs occur in the set of precedence relations is however unimportant.

The function `const()` uses a precedence relation pair. It checks whether the precedence relation of one node appearing before the other in the source path is also applicable to the candidate target path. When the candidate path has the nodes in the same order, `const()` returns 1, and otherwise it returns 0.

Using the preceding functions, the order-consistency can be computed using:

$$\text{orderConst}(P_{\text{src}}, P_{\text{targ}}) = \sum_{r \in R} \frac{\text{const}(r, P_{\text{targ}})}{\binom{\text{length}(C)}{2}} \qquad (14)$$

where P_{src} is the list of nodes from the source taxonomy path, P_{targ} is the list of nodes from a candidate target taxonomy path, C equals $\texttt{common}(P_{\text{src}}, P_{\text{targ}})$, R equals $\texttt{precedenceRelations}(C, P_{\text{src}})$, and r is a precedence relation. The denominator in this function, is the number of possible combinations of chosing two out of the common nodes. This makes the order-consistency the average number of precedence relations from the source taxonomy path that are consistent with a candidate target taxonomy path. Note that this measure is purely based on order. The number of co-occurring nodes is disregarded.

3.6. Selecting the Wining Candidate Target Category

Using the co-occurrence and order-consistency measures (Sections 3.4 and 3.5, respectively), a winner can be chosen from all candidate target paths. The Park & Kim algorithm computes the average of these measures, and take the candidate path with the highest score as winner. The proposed algorithm uses a different method to score each candidate target category, given by:

$$\text{sim}^*(P_{\text{src}}, P_{\text{targ}}) = (\text{orderConst}(P_{\text{src}}, P_{\text{targ}}) + t) \cdot \text{coOcc}(P_{\text{src}}, P_{\text{targ}}) \qquad (15)$$

where P_{src} is the list of nodes from the current source taxonomy path, P_{targ} is the list of nodes from a candidate target taxonomy path, and t is the similarity threshold. As shown in Equation (15), the algorithm makes use of the so-called similarity threshold. This threshold functions as a cut-off, to filter mappings that are based on a similarity that is insufficient. When the overall similarity between the source category and the best candidate target category is below the similarity threshold, the category will not be mapped by the algorithm.

The reason to use the similarity threshold in the overall similarity function, is the following: the order- consistency is in practice very often just 1 or 0. On one hand it does not occur often that paths are ordered differently; on the other hand, paths may not have overlapping nodes that can be found by the order- consistency procedure. That is for example the case with the highest level categories (below the taxonomy root), like 'Shopping $\rightarrow$ Entertainment' in Figure 4, in which 'Shopping' is the root. The order-consistency procedure will always return 0 in such a case, since no precedence relation pairs can ever be found, caused by the fact that there is only one category node that is used in the computation (the root is never used). In such cases, it can still be that a candidate path with an order-consistency of 0 is a good choice. To make sure that the overall similarity will be sufficient to pass the similarity threshold, the order- consistency is added to that threshold before multiplying with the co-occurrence. In this way, when a candidate path has a co-occurrence of 1, but an order-consistency of 0, it still has a chance to be used. By using the overall similarity function as described above, candidate top categories like 'Entertainment' still have a possibility to win.

Another observation from the overall similarity function, is that it uses a multiplication. The reason is to make the two similarity measures more dependent of each other in the overall similarity computation. With the same values for the co-occurrence and the order-consistency, the overall similarity is always lower than with taking the normal average. Candidates with reasonable scores for both measures have a much bigger chance of winning than those with a very low score for one and a very high for the other. With averages, these differences are much smaller, making the two measures more independent. Using a multiplication will, for example, prevent that categories that happen to have a high order-consistency, for instance, because of a low number of co-occurring nodes, are able to win.

In the end, the candidate target path with the highest overall similarity is chosen. If its overall similarity is higher than the similarity threshold, which is the only parameter in the algorithm, the mapping is executed. Otherwise, the source category is mapped to the same target category as its parent, according to one of the assumptions from Section 3.1. If the parent could not be mapped either, this becomes a mapping to nothing. The Park & Kim algorithm does not map to parent mappings. The complete process as described in this section, from word sense disambiguation, to candidate selection, to finding the best candidate, will then be restarted for the next path in the source taxonomy.

4. Evaluation

In this section, the proposed algorithm will be evaluated against the Park & Kim algorithm [16] and Anchor-PROMPT [20, 21]. Moreover, two versions of the proposed algorithm are evaluated separately, one using a word sense disambiguation process based on that of Park & Kim (Section 3.2.1), the other based on that of Lesk [55] (Section 3.2.2). These are respectively referred to as Aanen–Park and Aanen–Lesk in the tables in this section. Before the evaluation itself, the design of the evaluation experiment is discussed, together with the measures of evaluation.

4.1. Evaluation Design

This section discusses how the evaluation experiment has been set up. This includes the retrieval of evaluation data sets, data sampling, and the creation of the golden mappings.

Evaluation Taxonomies. For the evaluation of the proposed algorithm, three data sets were used. Since practical applications are possible with product taxonomy mapping, it is assumed that it is best to use real-life product taxonomies that are actually being used on the Web. The first taxonomy that is used, is from Amazon.com [71], which is the largest online retailer in the US, and one of the largest in the world. Amazon sells products for many different applications, which gives it a very wide taxonomy. These products are categorized in over 2,500 classes. The data was collected using a custom build HTML DOM crawler.

The second product taxonomy is from Overstock.com [72] (also referred to as O.co), a large retailer selling almost a million products, in over 1,000 categories. Overstock has tagged its product pages with some semantic descriptions, according to the GoodRelations [17] product ontology. These descriptions include the taxonomy structure, for which reason the data set of Overstock was collected using RDFa-crawlers for this research.

The last data set, is from ODP (or Dmoz) [73]. ODP (Open Directory Project, also known as Dmoz) is not a Web store, but a project with the aim to categorize the complete Web in a hierarchical structure. It is a multilingual collection of Web links, constructed and maintained by volunteers. While ODP is not only used for products, it does have a top category called 'Shopping'. Using an HTML DOM crawler, the taxonomy has been collected with 'Shopping' as root. This delivers more than a staggering 44,000 categories for products. It is both a very broad and deep taxonomy. ODP also contains very specific and sometimes unusual classes, which makes it a challenge for the matching algorithms to handle. The enormous size of the taxonomy puts the time performance of the algorithms to the test. The three data sets that are discussed, can all be used as source or target taxonomy. This results in six different combinations of mappings: from each of the data sets map to another. All six combinations are used in the evaluation.

Taxonomy Samples. With the three product taxonomies that are used for evaluation, six different mappings from source to target can be made. However, then the problem remains of how to specify for each of the mappings from category to category, whether it is correct or not, according to a human. This could be done either afterwards, or in advance of running the automated procedures. The assumption is, that creating a manual mapping beforehand is most objective, since all datasets will have to cope with the same specification of how the categories should be mapped. Also, by letting a human decide on correctness afterwards, extra bias can arise because sometimes a mapping option chosen by the algorithm might look good, while there actually exists a better option. Therefore, manual mappings are chosen to be used as reference for evaluation of the algorithm mappings.

The usage of manual mappings would require that for six combinations of data sets, all categories from the source taxonomy are linked to categories in the target taxonomy. Unfortunately, this would require too much manual labor. Therefore, random samples of five hundred categories have been taken for each data set. Manual mappings have been made with these samples as source taxonomies. The algorithms thus also always map from sampled data set to full data set. It was insured that for each category included in the random samples, the ancestors were also included. In this way, the taxonomies keep their tree structure. To reduce bias in the evaluation results, the manual mappings have been made by three individual people using majority voting.

Algorithm Mappings. For each of the six possible mappings between the taxonomies, the algorithms were executed. The algorithm of Park & Kim, and the proposed algorithm both have one parameter: the similarity threshold. Therefore, a batch of mappings has been executed for these algorithms. The similarity threshold has been chosen from 0 to 1 in steps of 0.05. For each mapping, the similarity threshold giving the best F_1-measure is shown in Tables 2, 3 and 4. The associated value is printed in the column 'Threshold'.

For every algorithm, the average computation times have also been measured. This is the average from the batches, making it independent of the similarity threshold. The times have been measured using a computer with an Intel Core 2 Duo E7600 processor at 3.06 GHz, 4 GB of memory, running Mac OS X. An overview of the computation times per mapping can be found in Table 5. The implementation of each algorithm was done using the Java programming language.

Table 1 shows the ratio of senses that have been found. This is based on the number of extended term sets that could be found by a word sense disambiguation procedure, against the number of times the algorithm asked to disambiguate something. Since the disambiguation procedures are only used to process category names from the source taxonomy, there are only three different possibilities (three taxonomies). The target taxonomy does not influence this result. Note that the ratio only explains how many of the senses are found, and not whether the best possible senses are found. The degree of correctness is expressed by the accuracy, which is stated in Table 7. The accuracy is measured by manually classifying each sense assigned by the word sense disambiguation processes to either be correct or incorrect. This has been done for all cases in which any sense could be assigned to a (split) category term from the sampled data sets. Note that this result is not very reliable in the case of the algorithm of Park & Kim or Aanen–Park, in combination with data sets from Amazon or Overstock, since the amount of cases in which a sense could be found

Table 2: Best results for the Park & Kim algorithm

Mapping	Precision	Accuracy	Specificity	Recall	F_1-measure	Threshold
Amazon → ODP	29.10%	20.80%	40.63%	11.47%	0.1645	0.10
Amazon → O.co	47.15%	31.80%	60.84%	17.37%	0.2539	0.00
ODP → Amazon	27.07%	42.48%	64.47%	15.93%	0.2006	0.05
ODP → O.co	31.89%	27.66%	38.54%	20.07%	0.2464	0.00
O.co → Amazon	54.29%	32.20%	45.21%	26.84%	0.3592	0.00
O.co → ODP	37.86%	26.60%	47.90%	15.92%	0.2241	0.00
Average	37.89%	30.26%	49.60%	17.93%	0.2415	

Table 3: Best results for the proposed algorithm with disambiguation process according to Section 3.2.1, based on Park & Kim

Mapping	Precision	Accuracy	Specificity	Recall	F_1-measure	Threshold
Amazon → ODP	25.37%	25.20%	13.65%	41.55%	0.3150	0.20
Amazon → O.co	55.85%	46.40%	42.08%	49.66%	0.5257	0.25
ODP → Amazon	26.63%	41.68%	54.08%	23.90%	0.2519	0.15
ODP → O.co	39.53%	31.86%	46.67%	22.37%	0.2857	0.10
O.co → Amazon	42.53%	34.00%	28.23%	38.14%	0.4022	0.20
O.co → ODP	25.15%	25.40%	17.44%	35.62%	0.2948	0.15
Average	35.84%	34.09%	33.69%	35.21%	0.3459	

is so small, that only approximately 10–20 cases could be evaluated on accuracy.

Performance Measures. The problem of mapping taxonomy classes is a classification problem. In classification, performance is usually evaluated using a confusion matrix. However, in this situation, there are not just two possible cases (false or true), but many different cases. We use the following definitions:

$$\text{true positives (TP)} = \text{\# mappings to the correct path}$$
$$\text{false positives (FP)} = \text{\# mappings to a faulty path}$$
$$\text{true negatives (TN)} = \text{\# correct null mappings}$$
$$\text{false negatives (FN)} = \text{\# incorrect null mappings}$$
$$\text{sensitivity or recall} = \frac{\text{TP}}{\text{P}} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$
$$\text{accuracy} = \frac{\text{TP} + \text{TN}}{\text{P} + \text{N}}$$
$$\text{specificity} = \frac{\text{TN}}{\text{N}} = \frac{\text{TN}}{\text{FP} + \text{TN}}$$

$$\text{precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$
$$F_1\text{-measure} = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

4.2. Results

Table 6 shows the mapping results for PROMPT. As becomes clear, PROMPT scores relatively bad for recall,

Table 4: Best results for the proposed algorithm with disambiguation process according to Section 3.2.2, based on Lesk

Mapping	Precision	Accuracy	Specificity	Recall	F_1-measure	Threshold
Amazon → ODP	31.45%	30.40%	16.61%	47.53%	0.3786	0.35
Amazon → O.co	53.02%	51.60%	23.83%	76.23%	0.6254	0.20
ODP → Amazon	20.23%	35.27%	37.05%	31.74%	0.2471	0.20
ODP → O.co	47.67%	37.07%	47.94%	30.16%	0.3695	0.25
O.co → Amazon	40.13%	35.40%	21.34%	48.28%	0.4383	0.20
O.co → ODP	21.97%	24.40%	14.56%	41.30%	0.2868	0.20
Average	35.75%	35.69%	26.89%	45.87%	0.3909	

Table 5: Average computation times per mapping for each algorithm (in seconds)

Mapping	PROMPT	Park & Kim	Aanen–Park	Aanen–Lesk
Amazon → ODP		12.76	19.67	83.79
Amazon → O.co		0.34	0.72	48.58
ODP → Amazon		0.86	1.52	65.39
ODP → O.co		0.39	0.68	60.55
O.co → Amazon		0.75	1.64	97.01
O.co → ODP		12.00	23.61	130.32
Average	0.47[a]	4.52	7.97	80.94

[a]Individual mapping times are not available for PROMPT (because it was run as a plugin in Protégé)

which is the ratio of mapped classes of the classes that should have been mapped. Moreover, the precision, which is the degree of correctness, is very poor: 19.82% on average. Here the fact that PROMPT is not specifically built for product taxonomy matching becomes clear. Product taxonomies have many characteristics that differ from general ontology or schema matching, such as polysemy, for which reason a customized approach is needed. PROMPT is extremely fast though, as Table 6 makes clear.

The approach from Park & Kim, of which the results are shown in Table 2, is specifically designed for product taxonomies. The combination of word sense disambiguation techniques to deal with polysemy, lexical similarity and path similarity, result in a much better performance than PROMPT. Worth mentioning is the relative good performance on precision. Nevertheless, the recall is rather poor, which means it is less suitable for e-commerce. This is seen as especially important in the e-commerce domain. Since generally very large amounts of data have to be processed, it is desirable that as many as possible categories will be handled, so that less human adaptations are required. Whether the categorization is exactly as wanted (the precision), is less important than making sure that every product is retrievable for search.

As Table 1 shows, the algorithm of Park & Kim is generally not very successful in disambiguating category names. For disambiguating categories from ODP the result is a lot better than for the other taxonomies. The explanation can be, that ODP is a very wide taxonomy, where classes are usually not (if not never) combined into composite categories. Park & Kim will for example try to disambiguate the term 'Electronics & Computers' as a

Table 6: Best results for AnchorPROMPT

Mapping	Precision	Accuracy	Specificity	Recall	F_1-measure
Amazon → ODP	7.74%	14.40%	29.56%	4.04%	0.0531
Amazon → O.co	35.59%	29.00%	57.54%	13.08%	0.1913
ODP → Amazon	8.08%	31.26%	43.48%	9.04%	0.0853
ODP → O.co	15.51%	20.84%	32.19%	10.90%	0.1280
O.co → Amazon	41.95%	27.80%	39.52%	21.92%	0.2879
O.co → ODP	10.07%	18.80%	39.02%	4.75%	0.0646
Average	19.82%	23.68%	40.22%	10.62%	0.1350

whole, which is very hard from a linguistic point-of-view. ODP however has separate categories, which are much easier to disambiguate. Furthermore, classes from ODP are often more specific, having more information content, than the more general classes from Amazon or Overstock.

Solving many of the problems from the Park & Kim algorithm, such as composite category handling, the proposed algorithm from this paper works better on average, as Table 3 makes clear. The average performance on recall increases by more than 17 percent point. The average computation time shown in Table 5 does increase a bit. This can be explained by the fact that more terms have to be disambiguated and handled, because of the splitting on composite categories. For the same reason, the ratio of senses that can be found by the word sense disambiguation process (Table 1), decreases slightly. Although the senses assigned to (split) categories are often correct, as Table 7 shows, the fact that on average no more than 5.7% of the senses can be found, undermines the need for a better disambiguation procedure.

The word sense disambiguation procedure explained in Section 3.2.2, addresses the desire of finding a larger ratio of senses. Table 1 shows that the disambiguation process based on [55] increases the ratio of senses that can be found by 622% on average, in comparison to the Park & Kim algorithm. Of all these senses that were assigned to category terms, 90.72% is correct on average. It is hard to tell whether the accuracy has changed in comparison to the disambiguation procedure of the Park & Kim algorithm, since that approach only finds a very small proportion of senses. The evaluation of the amount of those assigned senses that are correct is therefore less relevant. The fact that the ratio of found senses increases also results in better mapping performance (as depicted in Table 4). Due to the increase in recall, the overall performance, represented by the F_1-measure, increases significantly. However, this is at the cost of speed, as Table 5 shows. The overall computation time increases approximately by a factor of 10 on average. However, since taxonomies only have to be mapped once (or once in a while), we argue that this is not really an issue. On the other hand, it would be useful to speed up the disambiguation process because one could use more than two context words for the disambiguation. A larger context might improve the performance even more because it becomes more likely that the algorithm will find correct senses.

Table 7: Accuracy of senses found per source taxonomy for each algorithm

Data set	Park & Kim	Aanen–Park	Aanen–Lesk
Amazon	75.00%	94.12%	82.11%
ODP	96.15%	96.43%	97.50%
O.co	80.00%	83.34%	92.55%
Average	83.72%	91.30%	90.72%

As becomes clear in Table 5, all algorithms seem to struggle more with mappings to ODP as target, and mappings with Overstock as source. The reason that mapping to ODP is more time-consuming, is simply the enormous size. Over 44,000 categories have to be scanned to find candidate target categories for each path from the source taxonomy. Since Overstock has got a lot of composite categories, mapping with this data set as source also puts more stress on the time performance in the algorithms that handle this phenomenon. This is because the classes have to be split, after which disambiguation techniques are used for each part individually. In addition, candidate target category search is more complicated with split categories.

Table 8 gives an overview of the most important measures of the evaluation. From this table, we can see that our proposed algorithm performs best when used with the new word sense disambiguation procedure that is based on Lesk. We see that the precision of our approaches is similar to the Park & Kim algorithm, while our best approach achieves a recall that is almost twice as high as that of the Park & Kim algorithm.

5. Conclusion

In this paper we proposed an algorithm for the mapping of product taxonomies that are used in the e-commerce domain. This makes it possible to aggregate product information from multiple Web sites, in order to deploy it for comparison or recommendation applications. The evaluation results show that the proposed algorithm performs better than PROMPT [20, 21] and the Park & Kim algorithm [16]. The focus in this research was to improve the word sense disambiguation procedure from the Park & Kim algorithm. The approach based on Lesk [55] that we proposed, increases the amount of senses that can be found for category names by 622% on average, of which an average of 90.72% are correct. Since this comes at the cost of an increased computational cost, an improved version of the faster disambiguation procedure of Park & Kim was also discussed.

Furthermore, we proposed several other improvements to the Park & Kim algorithm. First, we now properly address the issue of composite categories, an issue not tackled by Park & Kim. By using the index of Jaccard [44] as similarity measure, instead of exact lexical matching, the problem of words occurring in different forms is also better dealt with.

We also found that the classification succeeds more often by mapping to more general target categories when more specific matches for a source category are not available. Overall, our approach showed an increase in F_1-measure by almost 62%, indicating the overall improvement of the proposed algorithm with disambiguation based on Lesk, in comparison to the approach of Park & Kim.

While the proposed algorithm improves existing approaches, there is still room for improvement. One of the drawbacks of the new word sense disambiguation procedure which prevents it from reaching its full potential, is that is only uses, for performance reasons, a context of two words: one parent category and one child category. One possible speed improvement could be to use a generalized Jaccard index for use with more than two sets, so a candidate combination of senses can be scored at once.

Furthermroe, instead of matching the glosses lexically, different (semantic) similarity measures could also be used. An example could be graph-based approaches, that measure the conceptual distance in a semantic lexicon. Such approach could improved accuracy by using all the information from the taxonmy at-once for deciding where to map particular categories.

The current order-consistency measure uses all synonyms of terms for matching, disregarding the sense. Desirably, this process should use only synonyms of the correct sense. Furthermore, the search for co-existing nodes in this process overlaps with that of the procedure to compute the co-occurrence measure. Ideally, the two measures

Table 8: Overview of average results per algorithm

Algorithm	Precision	Recall	F_1-measure	Avg. time	Senses found	WSD accuracy
PROMPT	19.82%	10.62%	0.1350	0.47 s	n/a	n/a
Park & Kim	37.89%	17.93%	0.2415	4.52 s	0.057	83.72%
Aanen–Park	35.84%	35.21%	0.3459	7.97 s	0.048	91.30%
Aanen–Lesk	35.75%	45.87%	0.3909	80.94 s	0.411	90.72%

should either be combined, or there should be an increase in the amount of information that is shared across the two methods.

Acknowledgment

Damir Vandic is supported by an NWO Mosaic scholarship for project 017.007.142: *Semantic Web Enhanced Product Search (SWEPS).*

References

[1] G.-Q. Zhang, G.-Q. Zhang, Q.-F. Yang, S.-Q. Cheng, T. Zhou, Evolution of the Internet and its Cores, New Journal of Physics 10 (2008) 123027.

[2] T. Berners-Lee, J. Hendler, O. Lassila, The Semantic Web, The Scientific American 284 (2001) 34–43.

[3] T. R. Gruber, A Translation Approach to Portable Ontology Specifications, Knowledge Acquisition 5 (1993) 199–199.

[4] N. F. Noy, M. A. Musen, Ontology Versioning in an Ontology Management Framework, Intelligent Systems, IEEE 19 (2004) 6–13.

[5] M. Hepp, P. De Leenheer, A. De Moor, Y. Sure, Ontology Management: Semantic Web, Semantic Web Services, and Business Applications, volume 7, Springer, 2007.

[6] L. Arlotta, V. Crescenzi, G. Mecca, P. Merialdo, Automatic Annotation of Data Extracted from Large Web Sites, in: International Workshop on Web and Databases 2003 (WebDB 2003), 2003, pp. 7–12.

[7] C. Bizer, J. Lehmann, G. Kobilarov, S. Auer, C. Becker, R. Cyganiak, S. Hellmann, DBpedia - A Crystallization Point for the Web of Data, Web Semantics: Science, Services and Agents on the World Wide Web 7 (2009) 154–165.

[8] D. Vandić, J. W. J. van Dam, F. Frăsincar, Faceted Product Search Powered by the Semantic Web, Decision Support Systems 53 (2012) 425–437.

[9] D. Benslimane, S. Dustdar, A. Sheth, Services Mashups: The New Generation of Web Applications, Internet Computing, IEEE 12 (2008) 13–15.

[10] J. B. Horrigan, Online Shopping, Pew Internet & American Life Project Report 36 (2008).

[11] E. Rahm, P. A. Bernstein, A Survey of Approaches to Automatic Schema Matching, The VLDB Journal 10 (2001) 334–350.

[12] N. F. Noy, Semantic Integration: A Survey of Ontology-Based Approaches, ACM SIGMOD Record 33 (2004) 65–70.

[13] H.-H. Do, S. Melnik, E. Rahm, Comparison of Schema Matching Evaluations, in: NODe 2002 Web and Database-Related Workshops, volume 2593 of *LNCS*, Springer, 2002, pp. 221–237.

[14] P. Shvaiko, J. Euzenat, A Survey of Schema-Based Matching Approaches, Journal on Data Semantics IV (2005) 146–171.

[15] Y. Kalfoglou, M. Schorlemmer, Ontology Mapping: The State of the Art, The Knowledge Engineering Review 18 (2003) 1–31.

[16] S. Park, W. Kim, Ontology Mapping between Heterogeneous Product Taxonomies in an Electronic Commerce Environment, International Journal of Electronic Commerce 12 (2007) 69–87.

[17] M. Hepp, GoodRelations: An Ontology for Describing Products and Services Offers on the Web, in: 16th International Conference on Knowledge Engineering: Practice and Patterns (EKAW 2008), volume 5268 of *LNCS*, Springer, 2008, pp. 329–346.

[18] C. Fellbaum, WordNet: An Electronic Lexical Database, The MIT press, 1998.

[19] J. H. Gennari, M. A. Musen, R. W. Fergerson, W. E. Grosso, M. Crubézy, H. Eriksson, N. F. Noy, S. W. Tu, The Evolution of Protégé: An Environment for Knowledge-Based Systems Development, International Journal of Human-Computer Studies 58 (2003) 89–123.

[20] N. F. Noy, M. A. Musen, The PROMPT Suite: Interactive Tools for Ontology Merging and Mapping, International Journal of Human-Computer Studies 59 (2003) 983–1024.

[21] N. F. Noy, M. A. Musen, Using PROMPT Ontology-Comparison Tools in the EON Ontology Alignment Contest, in: 3rd International Workshop Evaluation of Ontology-based Tools (EON 2004), 2004. `http://bit.ly/1dHCalY`.

[22] S. Castano, A. Ferrara, S. Montanelli, H-MATCH: An Algorithm for Dynamically Matching Ontologies in Peer-Based Systems, in: 1st VLDB Int. Workshop on Semantic Web and Databases (SWDB 2003), 2003, pp. 231–250. `http://bit.ly/14dl7nQ`.

[23] S. Castano, A. Ferrara, S. Montanelli, D. Zucchelli, HELIOS: A General Framework for Ontology-Based Knowledge Sharing and Evolution in P2P Systems, in: 14th International Workshop on Database and Expert Systems Applications (DEXA 2003), IEEE Computer Society, 2003.

[24] E. Mena, A. Illarramendi, V. Kashyap, A. P. Sheth, OBSERVER: An Approach for Query Processing in Global Information Systems Based on Interoperation across Pre-existing Ontologies, Distributed and Parallel Databases 8 (2000) 223–271.

[25] J. Li, LOM: A Lexicon-Based Ontology Mapping Tool, in: Performance Metrics for Intelligent Systems 2004 (PerMIS 2004), 2004. `http://bit.ly/1gQfYm2`.

[26] I. Niles, A. Pease, Towards a Standard Upper Ontology, in: International Conference on Formal Ontology in Information Systems 2001 (FOIS 2001), ACM, 2001, pp. 2–9.

[27] K. Kotis, G. A. Vouros, K. Stergiou, Towards Automatic Merging of Domain Ontologies: The HCONE-Merge Approach, Web Semantics: Science, Services and Agents on the World Wide Web 4 (2006) 60–79.

[28] S. Deerwester, S. T. Dumais, G. W. Furnas, T. K. Landauer, R. Harshman, Indexing by Latent Semantic Analysis, Journal of the American Society for Information Science 41 (1990) 391–407.

[29] D. L. McGuinness, R. Fikes, J. Rice, S. Wilder, The Chimaera Ontology Environment, in: 17th National Conference on Artificial Intelligence (AAAI 2000) and 12th Conference on Innovative Applications of Artificial Intelligence (IAAI 2000), AAAI Press, 2000, pp. 1123–1124.

[30] M. Ehrig, S. Staab, QOM - Quick Ontology Mapping, in: International Semantic Web Conference 2004 (ISWC 2004), 2004, pp. 683–697.

[31] I. Benetti, D. Beneventano, S. Bergamaschi, F. Guerra, M. Vincini, An Information Integration Framework for E-commerce, IEEE Intelligent Systems 17 (2002) 18–25.

[32] M. Ehrig, Y. Sure, Ontology Mapping — An Integrated Approach, in: 1st European Semantic Web Symposium. The Semantic Web: Research and Applications (ESWS 2004), volume 3053 of *LNCS*, Springer, 2004, pp. 76–91.

[33] A. Gangemi, N. Guarino, C. Masolo, A. Oltramari, L. Schneider, Sweetening Ontologies with DOLCE, in: 13th International Conference on Knowledge Engineering and Knowledge Management. Ontologies and the Semantic Web (EKAW 2002), volume 2473 of *LNCS*, Springer, 2002, pp. 223–233.

[34] S. Kaza, H. Chen, Evaluating ontology mapping techniques: An experiment in public safety information sharing, Decision Support Systems 45 (2008) 714–728.

[35] H.-H. Do, E. Rahm, COMA: A System for Flexible Combination of Schema Matching Approaches, in: 28th International Conference on Very Large Data Bases (VLDB 2002), VLDB Endowment, 2002, pp. 610–621.

[36] D. Aumueller, H.-H. Do, S. Massmann, E. Rahm, Schema and Ontology Matching with COMA++, in: ACM SIGMOD International Conference on Management of Data 2005 (SIGMOD 2005), ACM, 2005, pp. 906–908.

[37] S. Melnik, H. Garcia-Molina, E. Rahm, Similarity Flooding: A Versatile Graph Matching Algorithm and its Application to Schema Matching, in: 18th International Conference on Data Engineering (ICDE 2002), IEEE, 2002, pp. 117–128.

[38] J. Madhavan, P. A. Bernstein, E. Rahm, Generic Schema Matching with Cupid, in: 27th International Conference on Very Large Data Bases (VLDB 2001), Morgan Kaufmann Publishers Inc., 2001, pp. 49–58.

[39] F. Giunchiglia, P. Shvaiko, M. Yatskevich, S-Match: An Algorithm And An Implementation of Semantic Matching, in: Dagstuhl Seminar Proceedings of Semantic Interoperability and Integration 2005, 2005. `http://bit.ly/15m079i`.

[40] W.-S. Li, C. Clifton, SEMINT: A Tool for Identifying Attribute Correspondences in Heterogeneous Databases Using Neural Networks, Data and Knowledge Engineering 33 (2000) 49–84.

[41] B. He, K. C.-C. Chang, Automatic Complex Schema Matching across Web Query Interfaces: A Correlation Mining Approach, ACM Transactions on Database Systems 31 (2006) 1–45.

[42] B. Magnini, M. Speranza, C. Girardi, A Semantic-Based Approach to Interoperability of Classification Hierarchies: Evaluation of Linguistic Techniques, in: 20th International Conference on Computational Linguistics (COLING 2004), Association for Computational Linguistics, 2004, pp. 1133–1133.

[43] P. Avesani, F. Giunchiglia, M. Yatskevich, A Large Scale Taxonomy Mapping Evaluation, in: 4th International Semantic Web Conference (ISWC 2005), volume 3729 of *LNCS*, Springer, 2005, pp. 67–81.

[44] P. Jaccard, The Distribution of the Flora in the Alpine Zone, New Phytologist 11 (1912) 37–50.

[45] L. R. Dice, Measures of the Amount of Ecologic Association between Species, Ecology 26 (1945) 297–302.

[46] V. Levenshtein, Binary Codes Capable of Correcting Deletions, Insertions, and Reversals 10 (1966) 707–710.

[47] P. Resnik, Using Information Content to Evaluate Semantic Similarity in a Taxonomy, in: 14th International Joint Confer-

ence on Artificial Intelligence (IJCAI 1995), Morgan Kaufmann, 1995, pp. 448–453.

[48] S. Patwardhan, S. Banerjee, T. Pedersen, Using Measures of Semantic Relatedness for Word Sense Disambiguation, in: 4th International Conference on Computational Linguistics and Intelligent Text Processing (CICLing 2003), 2010, pp. 241–257.

[49] J. J. Jiang, D. W. Conrath, Semantic Similarity Based on Corpus Statistics And Lexical Taxonomy, CoRR cmp-lg/9709008 (1997). `http://bit.ly/18xmEkv`.

[50] D. Lin, Using Syntactic Dependency as Local Context to Resolve Word Sense Ambiguity, in: 35th Annual Meeting of the Association for Computational Linguistics (ACL 1997) and 8th Conference of the European Chapter of the Association for Computational Linguistics (EACL 1997), Association for Computational Linguistics, 1997, pp. 64–71.

[51] C. Leacock, M. Chodorow, Filling in a Sparse Training Space for Word Sense Identification, March, 1994.

[52] D. Lin, An Information-Theoretic Definition of Similarity, in: 15th International Conference on Machine Learning (ICML 1998), Morgan Kaufmann Publishers Inc., 1998, pp. 296–304.

[53] A. Hogenboom, F. Hogenboom, F. Frasincar, K. Schouten, O. van der Meer, Semantics-Based Information Extraction for Detecting Economic Events, Multimedia Tools and Applications 64 (2012) 27–52.

[54] B. Heerschop, F. Goossen, A. Hogenboom, F. Frasincar, U. Kaymak, F. de Jong, Polarity Analysis of Texts using Discourse Structure, in: B. Berendt, A. de Vries, W. Fan, C. Macdonald, I. Ounis, I. Ruthven (Eds.), Twentieth ACM Conference on Information and Knowledge Management (CIKM 2011), Association for Computing Machinery, 2011, pp. 1061–1070.

[55] M. Lesk, 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), ACM, 1986, pp. 24–26.

[56] S. Banerjee, T. Pedersen, An Adapted Lesk Algorithm for Word Sense Disambiguation using WordNet, in: 3rd International Conference on Computational Linguistics and Intelligent Text Processing (CICLing 2002), 2002, pp. 136–145.

[57] A. Kilgarriff, J. Rosenzweig, Framework and Results for English SENSEVAL, Computers and the Humanities 34 (2000) 15–48.

[58] F. Vasilescu, P. Langlais, G. Lapalme, Evaluating Variants of the Lesk Approach for Disambiguating Words, in: 4th Conference of Language Resources and Evaluations (LREC 2004), 2004, pp. 633–636.

[59] E. Agirre, G. Rigau, Word Sense Disambiguation using Conceptual Density, in: 16th International Conference on Computational Linguistics (COLING 1996), 1996, pp. 16–22.

[60] D. Yarowsky, Word-Sense Disambiguation Using Statistical Models of Roget's Categories Trained on Large Corpora, in: 14th International Conference on Computational Linguistics (COLING 1992), 1992, pp. 454–460.

[61] C. Leacock, G. A. Miller, M. Chodorow, Using Corpus Statistics and WordNet Relations for Sense Identification, Computational Linguistics 24 (1998) 147–165.

[62] R. Navigli, P. Velardi, Structural Semantic Interconnections: A Knowledge-Based Approach to Word Sense Disambiguation, IEEE Transactions on Pattern Analysis and Machine Intelligence 27 (2005) 1075–1086.

[63] R. Navigli, Word Sense Disambiguation: A Survey, ACM Comput. Surv. 41 (2009) 10:1–10:69.

[64] SemCor, Sense-Tagged Corpus for Evaluation of Disambiguation Algorithms, `http://bit.ly/172YTme`, 2013.

[65] Senseval (or SemEval), Conference on Word Sense Disambiguation Algorithm Evaluation, `http://www.senseval.org/`, 2013.

[66] G. A. Miller, WordNet: A Lexical Database for English, Communications of the ACM 38 (1995) 39–41.

[67] E. Pianta, L. Bentivogli, C. Girardi, Developing An Aligned Multilingual Database, in: 1st International Conference on Global WordNet (GWC 2002), 2002.

[68] P. Vossen, EuroWordNet A Multilingual Database with Lexical Semantic Networks, Computational Linguistics 25 (1998).

[69] D. Tufis, D. Cristea, S. Stamou, BalkaNet: Aims, Methods, Results And Perspectives. A General Overview, Science and Technology 7 (2004) 9–43.

[70] G. de Melo, G. Weikum, Towards a Universal Wordnet by Learning from Combined Evidence, in: 18th ACM Conference on Information and Knowledge Management (CIKM 2009), ACM, 2009, pp. 513–522.

[71] Amazon.com, US' largest online retailer, `http://www.amazon.com`, 2013.

[72] Overstock.com, RDFa-annotated web store, `http://www.o.co`, 2013.

[73] ODP (or Dmoz), Open Directory Project, `http://www.dmoz.org/`, 2013.