Journal of Web Engineering, Vol. 15, No. 1&2 (2016) 084–109
© Rinton Press

A LEXICAL APPROACH FOR TAXONOMY MAPPING

LENNART NEDERSTIGT, DAMIR VANDIC, and FLAVIUS FRASINCAR
Econometric Institute, Erasmus University Rotterdam
P.O. Box 1738, 3000 DR Rotterdam, the Netherlands
lennart@grible.co, vandic@ese.eur.nl, frasincar@ese.eur.nl

Received April 29, 2015
Revised September 30, 2015

Obtaining a useful complete overview of Web-based product information has become difficult nowadays due to the ever-growing amount of information available on online shops. Findings from previous studies suggest that better search capabilities, such as the exploitation of annotated data, are needed to keep online shopping transparent for the user. Annotations can, for example, help present information from multiple sources in a uniform manner. In order to support the product data integration process, we propose an algorithm that can autonomously map heterogeneous product taxonomies from different online shops. The proposed approach uses word sense disambiguation techniques, approximate lexical matching, and a mechanism that deals with composite categories. Our algorithm's performance compared favorably against two other state-of-the-art taxonomy mapping algorithms on three real-life datasets. The results show that the F_1-measure for our algorithm is on average 60% higher than a state-of-the-art product taxonomy mapping algorithm.

Keywords: Schema mapping, product taxonomies, lexical matching, word sense disambiguation
Communicated by: G.J. Houben & O. Diaz

1 Introduction

It has become easier to share information on the Web. As a result of this, the Web doubles in size roughly every five years [36]. Because of this ever-growing amount of information, several problems arise. One of the problems is that it is becoming increasingly difficult to get a proper overview of all the relevant Web information. While traditional search engines use indexing to solve this problem, they do not understand the actual information on Web pages. This is due to the fact that most Web pages are geared towards human-readability, rather than machine-understandability. Humans, unlike machines, are able to extract the meaning of words from the context of Web pages. This is particularly a problem when searching using words that can have multiple meanings, like 'jaguar', which can either be a car or an animal. Most search engines will include every page that contains the search term in the search results, regardless of whether it is actually relevant or not.

The problem of query ambiguity is also manifested in e-commerce, as nowadays there are many products and shops available on the Web. The findings of a study on online shopping in the USA [14] suggest that better search functionalities are needed in order to keep online shopping transparent for the user. More than half of the respondents had encountered difficulties while shopping online. Users often found information to be lacking, contradictory, or overloaded. This emphasizes the need to aggregate the information found in those Web shops and presenting it in a uniform way.

While some price comparison sites, such as [34], do show aggregated information, they are often restricted in their use, as they only include regional price information and therefore compare only a limited number of online shops [37]. Furthermore, in order to be included in the comparison, online shops have to take the initiative and provide their data in a specific format that is defined by each price comparison site. This is a laborious task as the online shops have to export their data into different formats (there is no standardized semantic format for exchanging information). Consequently, product information sharing on the Web is often not done as it requires a significant amount of manual work.

A possible solution to the search problems encountered on the Web would be to annotate the data found on Web pages using standardized ontologies. In this way, the data becomes understandable to computers and product information sharing becomes significantly easier. For e-commerce, there is a well-established ontology available, called GoodRelations [13]. Unfortunately, not that many Web pages have so far included a semantic description for their content. Furthermore, even when a semantic description is available, not every Web page will use the same ontology. That is why there is a need for algorithms that are able to (semi-)automatically map product ontologies to each other.

Taxonomies can be seen as the basis of an ontology, as they contain the type-of relations. In a previous study [1], we have proposed a semantic approach for automated taxonomy mapping, extending the approach of [32], and focusing on improving the word sense disambiguation process. In this paper, we propose a taxonomy mapping algorithm, also based on the approach of [32], that focuses on advanced lexical matching. The proposed algorithm autonomously provides mappings between heterogeneous product taxonomies coming from multiple sources. Similar to the Park & Kim algorithm, our algorithm employs word sense disambiguation techniques to find the correct sense of a term using the semantic lexicon WordNet [25]. Differently to the Park & Kim algorithm, our algorithm considers for lexical matching various lexical similarity measures, like the Jaccard index and the Levenshtein distance. Our proposed algorithm also exploits the hierarchical structure of taxonomies by taking into account the distance between each candidate path and previous mappings. In addition, a different similarity aggregation function is used to make the algorithm more robust against outliers.

The contribution of this paper relates to several areas. First, we outperform the Park and Kim algorithm w.r.t. the F_1-measure by improving several aspects of the algorithm. For example, we use a more advanced lexical matching process. Second, we perform the evaluation of the algorithms on a much larger and more heterogeneous data set than in [32]. Consequently, our reported results are more reliable and generalizable. A short description of the proposed algorithm has been presented in [26]. With respect to this previous publication, we explain the algorithm in more details in this paper. Furthermore, the evaluation in this paper is more complete, i.e., we discuss the results in more depth and cover aspects that were not included in the previous publication. For example, we analyze the performance characteristics of the algorithms w.r.t. the different data sets used in the evaluation (instead of reporting only aggregated results).

2 Related work

In this section, we discuss a number of mapping algorithms that are related to, or can be used in the context of, product taxonomy mapping. Taxonomy mapping can be important for several use-cases, for example, recommender systems [38, 20]. We do not present methods that focus solely on entity disambiguation, although some of the techniques employed for this purpose can be re-used. A recent study on this topic can be found in [27].

In general, we can make a distinction between ontology mapping algorithms and schema mapping algorithms. Ontology mapping algorithms map hierarchical structures, which are often represented in OWL or RDF format. These formats have defined formal semantic relations between classes, which can be exploited when mapping ontologies. Schema mapping algorithms map relational schemas, such as relational databases and XML Schema, which do not possess the formal semantics that ontologies have. The mapping of the hierarchical structure of ontologies, without considering its instances and the formal relations between nodes, is referred to in related work as schema mapping. In order to avoid this ambiguity, with schema mapping we refer solely to the mapping of relational schemas, whereas the mapping of an hierarchical structure will be referred to as taxonomy mapping.

2.1 Ontology Mapping Algorithms

The algorithm proposed by [32] focuses on product taxonomy mapping in particular, rather than ontology mapping in general. Due to this focus, it achieves a higher recall than more general approaches, such as Anchor-PROMPT, when mapping product taxonomies [32]. General approaches only map when the similarity between two classes is relatively high. However, product taxonomies are arbitrarily made, which makes mapping them a loosely defined domain. Furthermore, the mapping of product taxonomies is aimed at reducing search failures when shopping online, and it is thus better to map more classes, even when two classes are not completely similar.

The algorithm starts by employing *word sense disambiguation* techniques to determine the sense of a category term from one of the taxonomies. It uses WordNet [25] to find the possible senses of the term, as well as the hypernyms of each of those senses. The hypernym structure of each sense is then compared with the ancestor nodes of the source category and the sense with the largest similarity between its hypernym structure and the ancestor nodes is chosen. Using the synonyms associated with the correct sense, the algorithm identifies candidate paths in the target taxonomy to which the current category can be mapped. It picks the best candidate by computing a similarity score for each candidate path. This similarity score consists of two functions, called co-occurrence and order-consistency. Co-occurrence measures the amount of overlap between categories in source and target paths, while order-consistency measures the degree to which categories in both paths occur in the same order. In the end, the algorithm maps to the best candidate path, provided that its similarity score is equal to or greater than a user-configurable threshold.

While the above algorithm is suitable for product taxonomy mapping, it has some points that could be improved. For instance, the algorithm does not consider the existence of composite categories, which are categories that consist of multiple concepts, like 'Movies, Music & Games'. Mapping these categories often fails, because the *word sense disambiguation* process does not work for these categories. This is due to the fact that it is unable to find the sense of the whole string in WordNet. Furthermore, it has difficulties disambiguating categories with short paths to the root, because of their lack of information content. This could be improved by also considering children and siblings of a category node when disambiguating. Another drawback of the algorithm is its bias towards mapping to short paths, which are sometimes too general. For example, consider a composite category called 'Movies, Music & Games', which has a subcategory called 'Music'. When mapping a category called 'Music', the algorithm will choose to map to the more general composite category, as the algorithm assumes that more general categories are more likely to be correct.

PROMPT [30], which stands for Protégé Ontology Management Plug-in Tool, is a plug-in for the open-source and freely available Protégé ontology management framework [9]. It provides a set of tools capable of managing ontologies, which can compare different versions of an ontology, merge ontologies, and map different ontologies. It has an interactive ontology mapping tool, called iPROMPT, which guides a user through the mapping process. The iPROMPT algorithm provides a set of initial mapping suggestions, based on the lexical similarity between nodes (i.e., node labels), from which the user can choose. The algorithm performs quite well on recall and precision [30], but it requires a lot of manual labor and it is therefore not usable for mapping large ontologies, which is not desired in this case.

The PROMPT suite also contains an extension of iPROMPT, called Anchor-PROMPT. This algorithm provides a (semi-)automatic ontology mapping process. It starts with a set of previously defined mappings. These mappings can be either specified by the user or provided by the algorithm using lexical term matching. The mappings from this set are then used as additional points of similarity, called *anchors*. Using these *anchors*, Anchor-PROMPT automatically generates a larger set of mappings by analyzing the graph structure between *anchors* and identifying concepts that appear at similar places in the ontologies. Because the performance of AnchorPROMPT largely depends on the accuracy of the initial mappings that are provided, it is not suitable for fully automatic ontology mapping [30], as the creation of initial mappings is a time-consuming task.

In, [19] LOM is proposed, which stands for Lexicon-based Ontology Mapping. It is a semi-automatic ontology mapping tool that suggests how to map the ontology vocabularies, while leaving the final decision to the user. It uses four stages, each with an increasing complexity, to determine similar concepts. Only the concepts from the source ontology that are left unmapped in the previous stage will be analyzed by the next stage. The first part of the algorithm simply uses exact string matching to compare whole terms. The second stage breaks up a whole term into separate words, while also removing stop words, such as 'and'. Furthermore, some morphological processing is used to get the lemma of each word. Those lemmas are then compared using exact string matching. The third stage looks up the lemmas in the WordNet semantic lexicon [25], which returns sets of synonyms, called synsets, of each lemma. The pairs of lemmas that have the largest ratio of common synsets between them are considered to be the best mapping. The final stage returns the mappings between these synsets and their associated concepts in formal ontologies, such as SUMO [28] and MILO [29]. Because this algorithm does not take the hierarchical structure of the ontologies into account, it is not considered to be useful for product taxonomy mapping.

The authors of [8] propose QOM, which stands for Quick Ontology Mapping. It was designed as a trade-off between the quality of a mapping and the speed with which a mapping can be made. This is especially useful for mapping large ontologies, such as SUMO [28], where more sophisticated approaches are deemed too inefficient to be used effectively. QOM uses heuristics to narrow down the number of potential candidate mappings. It computes similarities between two elements, by using an aggregated similarity score. This aggregated similarity score consists of the similarity between the represented concepts, relations, instance and instance properties. It is calculated using strict string matching, as well as the Levenshtein distance, and the Dice coefficient. In order to speed up the complete mapping, the algorithm only considers the most promising mappings in each iteration of the algorithm, which evaluates only a subset of the candidate mappings. This prevents the algorithm from having to compare a large number of nodes each time. While the focus of this algorithm on speed can be considered useful for mapping product taxonomies, the trade-off between speed and quality is not favorable within this domain. Also, because product taxonomies can become quite large, they are much smaller than the large ontologies for which this algorithm was designed. Therefore, the need to reduce the runtime complexity of the algorithm is considered less important for smaller ontologies, and a better accuracy of the mappings is preferred instead. Furthermore, a large part of its aggregated similarity score depends on the information provided by the annotation of the data in a formal ontology, which has well-defined relations and labels. Product taxonomies typically do not contain this extra information, which makes this algorithm unsuitable for mapping product taxonomies.

In [3], COMA++ is proposed, an automated mapping tool that can perform both schema and ontology mapping. It is an extended version of COMA [7]. It uses a composite approach to combine different matching algorithms. It is able to perform match iterations, where each match iteration can be individually configured to use different measures or perform matching on only a fragment of the data. The mapping output of each iteration can be used as an input for the next match iteration. This provides a large degree of flexibility, which allows the algorithm to be fine-tuned for each specific mapping problem. Furthermore, it can also map two schemas or ontologies to each other through the use of a *pivot schema*, which is a central schema or ontology for a specific domain. By mapping both schemas or ontologies to the *pivot schema*, one can enhance the mapping between the two schemas or ontologies. The use of a *pivot schema* might also prove beneficial for mapping product taxonomies, as it could help to improve the mappings between two product taxonomies. Because product taxonomies are arbitrarily made, the resulting hierarchy might not be well-structured. By using mappings to a standard product taxonomy, with a proven well-structured hierarchy, the mappings between two product taxonomies can be enhanced in this way. However, a standard product taxonomy does not yet exist, which means that this approach cannot be applied immediately.

The aforementioned algorithms are just a few of the algorithms that deal with ontology mapping. Chimaera [23] can be used as a standalone semi-automatic ontology mapping algorithm or used within the Ontolingua framework [11]. It gives suggestions for potential mappings by considering lexical similarity between elements as well as taking structural similarity into account, by analyzing semantic subsumption relationships between ontology elements. H-Match [4], which is part of the HELIOS framework [5], takes the taxonomy context as well as the linguistic affinity between ontology elements into account. In this approach WordNet is used as a thesaurus. An extensive survey of various ontology mapping algorithms can be found in [16].

2.2 Schema Mapping Algorithms

The authors of [24] propose an algorithm to semi-automatically map schemas, using an iterative fixpoint computation, which they dubbed *similarity flooding*. First, the schemas are converted to directed labeled graphs. Then a set of initial similarities between nodes in the graphs is generated by measuring their lexical similarity. The algorithm assumes that whenever two elements in the graphs are found to be similar, the similarity of their adjacent elements also increases. Therefore, by performing several iterations, the initial similarity between two nodes is propagated throughout the graphs. This will continue until the similarities stabilize. Finally, a threshold is selected, which functions as a cut-off for too weak similarities between nodes. Every pair of nodes in the two graphs which exceeds the threshold, will be considered similar and is thus mapped. In the final stage, the user examines the mappings and corrects them. While this approach could be applied to product taxonomies, it is not ideal. Only the lexical similarity is considered for the initial similarity, while semantic similarity is also important in product taxonomies. Furthermore, it does not scale well for large product taxonomies, which easily consist of thousands of categories, as the manual inspection of the mappings is time-consuming.

Cupid [21] is a general-purpose schema matching algorithm. It is a hybrid matcher, which exploits both lexical and semantic similarities between elements, while also comparing data types and constraints between elements. In addition, it also analyses the schema structure. The element-based similarity is calculated by tokenizing the terms first, which splits a term in individual words and removes punctuation marks. It also expands known abbreviations and acronyms, using a thesaurus. In addition, the elements are categorized and clustered together, according to the concept to which they belong. Only the elements that belong to 'compatible' categories are compared. Categories are considered compatible when the linguistic similarity between the contained concepts exceeds a defined threshold. Hypernyms and synonyms of the concept are also used for computing this similarity. The linguistic similarity between two individual elements from two categories is computed using the same function, which is then scaled by the similarity between their respective categories to obtain an overall similarity between elements. Using the similarity between elements, the structural similarity can be computed. It takes the similarity between individual elements into account, as well as their data types. Because a top-down approach is considered too optimistic and error-prone when the two schemas differ considerably at the top level [21], a bottom-up approach is used to compute the similarity between graphs.

There are various schema matching algorithms that exploit the data types and constraints of a relational database in order to map them. An example of an algorithm that uses this information is S-Match [10]. There are also entirely different approaches to schema mapping, i.e., ones that use machine learning techniques from the artificial intelligence domain. Examples of such algorithms are given in [22] and [12]. Extensive surveys of various schema mapping algorithms can be found in [33] and [35].

3 Schema Mapping Algorithm

In this section, we discuss the proposed approach and compare it against existing solutions. In particular, we focus on the differences between our approach and the one presented in [32], which we use as a foundation for our algorithm. First, we give a high level overview of our proposed approach. Then, we discuss the product taxonomy context and the assumptions that we make. Last, we go into the details of all steps for our proposed algorithm.

3.1 Overview

Our algorithm requires two inputs, a source taxonomy and a target taxonomy. The mapping is performed from the source taxonomy to the target taxonomy. Figure 1 illustrates the various processes that form the framework of both algorithms. The entire process is repeated for every source category that needs to be mapped and its associated path in the source taxonomy. Rounded rectangles represent the processes and the ovals represent the output. The processes and output that are colored gray (in a rectangle for black and white printing) are used in our algorithm, but not in the Park and Kim algorithm. The other processes and outputs occur in both algorithms.

Our algorithm starts with several pre-processing steps of the term found in a node of the path from the source taxonomy. It splits the term based on ampersands, commas, and the word 'and', resulting in a set that contains multiple terms. We refer to this set as the *split term set*. This step is performed to enhance the *word sense disambiguation* process for composite categories, which are categories that represent multiple concepts.

The first process in both algorithms is the word sense disambiguation process. This process aims to determine the correct meaning of the term in the leaf node by looking up the term in WordNet, a semantic lexicon [25]. The correct sense of the term can be found by comparing the hyponyms of each sense found in WordNet with all the ancestor nodes in the path of the source taxonomy. Our algorithm repeats this process for each term in the *split term set*. The result of this process is the *extended term set*, which contains the original term, and, if it was able to determine the correct sense of the term, also the synonyms of the term. Because our algorithm splits the original term into multiple split terms, we define the *extended split term set* as the set of all extended term sets (each corresponding to one split term).

Using the extended (split) term set obtained from the word sense disambiguation process, the algorithm analyzes the target taxonomy and determines which paths are considered as candidate paths for the mapping of the path from the source taxonomy. This is achieved by searching for paths that contain at least half of the terms in the extended (split) term set.

In order to determine which of the candidate paths is the best path to map to, both algorithms compute the *co-occurrence* and *order-consistency* for each path. The *co-occurrence* represents the level of overlap between the source taxonomy path and one of the candidate target paths, while disregarding the hierarchy. The *order-consistency* is essentially the ratio of common nodes, i.e., nodes occurring in both source path and candidate path, but in the same hierarchical order. Our algorithm uses a third measure, the *parent mapping distance*, which is the normalized distance in the target taxonomy between a candidate path and the path to which the parent in the source path was mapped to.

Using the similarity measures obtained in the previous step, the algorithms determine the best target path to map the source path to. While Park and Kim use the arithmetic mean of the co-occurrence and order-consistency, our algorithm uses the harmonic mean of the co-occurrence, the order-consistency, and the parent mapping distance. The path with the highest mean is then selected as the path to map to, provided that the mean is at least equal to a configurable threshold. If it fails to reach the threshold, or if no candidate paths were

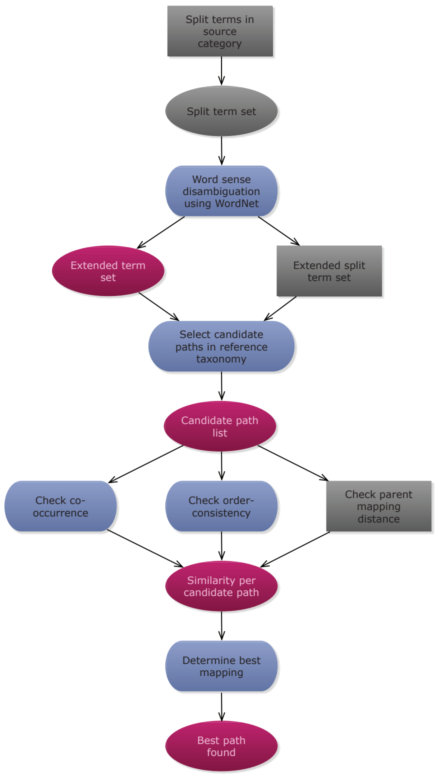
Split terms in source category
Split term set
Word sense disambiguation using WordNet
Extended term set
Extended split term set
Select candidate paths in reference taxonomy
Candidate path list
Check co-occurrence
Check order-consistency
Check parent mapping distance
Similarity per candidate path
Determine best mapping
Best path found

Fig. 1. Overview of the processes for our algorithm and the algorithm proposed by Park and Kim. Gray processes (in a rectangle for black and white printing) represent our contributions.

found, the Park and Kim algorithm simply does not map the source path. In this situation, our algorithm maps the source path to the same path in the target taxonomy to which its parent is mapped. If the parent of the source path was not mapped, our algorithm also does not map the source path.

3.2 Product taxonomy aspects

When analyzing product taxonomies, one can observe that virtually every taxonomy has a root category term without a meaning. For example, the root nodes from the datasets used in this research are 'Online Shopping', 'Shopping', and 'Products'. Therefore, these root nodes add nothing to the matching process and should therefore be discarded. For this reason, our algorithm automatically maps the root node from the source taxonomy to the root node of the target taxonomy. Furthermore, it skips the root node in all the computations that use category paths. In other words, the algorithm prevents the root nodes from polluting the search for candidate paths and the similarity between source and reference paths. The assumption that root nodes are meaningless is not made by Park & Kim, who do take the root node into account.

Another important phenomenon, which is neglected by Park and Kim, is the fact that a single category in a taxonomy might actually represent multiple categories, all clustered into this one category. We call such a category a *composite category*. For example, the

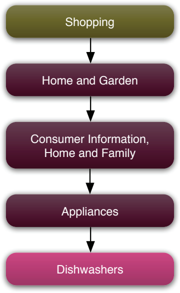

Fig. 2. Hierarchy for 'Dishwashers' in the ODP taxonomy

category 'Home, Garden & Tools', which occurs in the taxonomy of Amazon.com, represents the concepts 'Home', 'Garden', and 'Tools'. It is clear that these three concepts all represent a different kind of product, yet Park and Kim consider it as one category.

Composite categories pose a problem when trying to find synonyms in the word sense disambiguation process, as WordNet is unable to find synonyms for the whole category term. However, WordNet is likely to find synonyms for the individual terms. Furthermore, it is likely that online shops will use different composite categories and also in different locations in the taxonomy. By using the whole category term for searching candidate paths in the target taxonomy, it is unlikely that one finds a category that precisely matches the composite category. For these reasons, our algorithm performs several processing steps on category terms in both taxonomies. It splits the original category term into multiple terms whenever it encounters an ampersand, comma, or the word 'and'. We should also note that all term matching within the algorithm is case-insensitive, since the case of a letter generally does not affect the meaning of a word.

3.3 Word Sense Disambiguation

As previously discussed in Section 3.2, our algorithm splits composite categories and puts the individual terms in a set of terms, called the 'split term set'. This means that rather than using the entire category term for the word sense disambiguation process, like the Park and Kim algorithm does, it performs this process separately for each term in the 'split term set'.

Both algorithms enhance their ability to perform a correct mapping by first trying to determine the correct sense of a category term from the source taxonomy. This is useful because it helps to identify semantically similar categories from different taxonomies, even when they are not lexically similar. For instance, if the path from the source taxonomy is 'Computers/Notebook', we can deduce that the correct sense would be a laptop in this case, rather than a notepad. We could then include the word 'laptop' in our search terms used for identifying candidate paths, which might yield better candidate paths than only searching for 'notebook'. Being able to match semantically similar concepts, even when they are lexically dissimilar, is important when matching product taxonomies. Because there is no convention for category names, each taxonomy might be using different words to express the same product category.

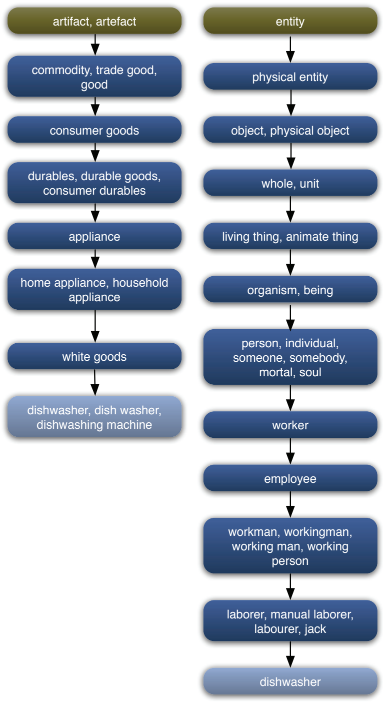
artifact, artefact
commodity, trade good, good
consumer goods
durables, durable goods, consumer durables
appliance
home appliance, household appliance
white goods
dishwasher, dish washer, dishwashing machine
entity
physical entity
object, physical object
whole, unit
living thing, animate thing
organism, being
person, individual, someone, somebody, mortal, soul
worker
employee
workman, workingman, working man, working person
laborer, manual laborer, labourer, jack
dishwasher

Fig. 3. Two sense hierarchies for the term 'dishwasher' in WordNet

3.3.1 WordNet

Similar to the approach of Park and Kim, our approach uses WordNet [25] to obtain all possible senses (i.e., meanings) of a category term from the source taxonomy. In order to find the correct sense, it has to match the full path of a category against the hierarchy of hypernyms for each sense from WordNet. Just to make it clear: the full path means the entire path besides the root node for our algorithm, whereas the full path for the Park and Kim algorithm does include the root node. Figure 2 shows an example of a path from a source taxonomy, where 'Dishwashers' is the category term for which we want to obtain all possible senses. Upper categories use dark purple (gray in black and white printing) nodes in this figure. We look up this category term in WordNet and obtain the different meanings of the word in the form of their hypernym structure, which is referred to as the *sense hierarchy*. Figure 3 shows the two senses that it finds for the term 'dishwashers'. The synonyms of 'dishwashers' for each sense are shown at the bottom in light blue (light gray in black and white printing).

The goal of the word sense disambiguation process is to obtain only one set of these synonyms, namely the one for the correct sense. Together with the original term, these synonyms form the 'extended term set', which will be used for finding candidate paths in the target taxonomy later on. Note that for the word sense disambiguation process only the source taxonomy is used and the target taxonomy does not play a role yet.

3.3.2 Matching a WordNet Sense

In order to find the meaning that fits most closely to the source category that needs to be mapped, a function is needed that can identify matches between an upper category, i.e., an ancestor of the current node from the source taxonomy, and a sense hierarchy obtained from WordNet, which is denoted by list S. Each sense hierarchy contains one meaning of the source category that is currently examined. To illustrate this aspect, Figure 3 shows the two sense hierarchies obtained from WordNet when looking up the category term 'Dishwashers'. The function for finding the matching terms is given by:

$$\mathrm{sim}(t, S) = \{x | x \in H, H \in S \text{ and } \mathrm{baseform}(t) \in H\} \tag{1}$$

where t is an upper category from the source taxonomy, H is the set of synonyms of a hypernym, and S is a sense hierarchy of a sense obtained from WordNet. Because the correct sense has not been found yet, the `sim` function takes also all the synonyms of the wrong sense into account. The `baseform()` function performs morphological processing on the term and returns the lemma of the term from WordNet. This is needed to filter out syntactical variations and convert plural words to singular, which is needed for correctly identifying the similarity between the upper categories from the source taxonomy and the sense hierarchies obtained from WordNet. The result of `sim()` is a set of matching lemmas between an upper category and a sense hierarchy, which represents one meaning of the current source category.

3.3.3 Source Category Ancestor WordNet Similarity

Using the set of matching lemmas, obtained from Equation 1, we can measure how well each upper category of the source taxonomy fits to each sense hierarchy. This is done by comparing each upper category with all the sense hierarchy nodes that are in the set. Because the information content per node in a sense hierarchy increases when a node is closer to the leaf, we try to find the match with the shortest distance to the sense hierarchy leaf. The similarity score, called the *hyperproximity*, increases when this distance is shorter, and is given by:

$$\mathrm{prox}(t, S) = \begin{cases} 1/\min\limits_{x \in C}(\mathrm{dist}(x, \ell)) & \text{if } C \neq \emptyset \\ 0 & \text{if } C = \emptyset \end{cases} \tag{2}$$

where t is an upper category to match, S is a sense hierarchy (one sense from WordNet), C is set to $\mathrm{sim}(t, S)$, and ℓ is the leaf of the sense hierarchy S. The `dist()` function computes the distance between each matching lemma in set x and the leaf node ℓ in the sense hierarchy. The distance is given by the number of edges that are traversed when navigating from the node with the matching lemma to the leaf node in the sense hierarchy. The `min()` function then selects the shortest distance that is returned by `dist()`.

For example, the `prox()` function returns 1/3 when matching the upper category 'Appliances' in Figure 2 with the left sense hierarchy in Figure 3. It matches with the sense hierarchy node 'appliance', which has a distance of 3 between it and the sense hierarchy leaf node.

3.3.4 Source Path WordNet Similarity

Now that we have the hyperproximity between each upper category from a source path and a particular sense hierarchy from WordNet, we can calculate the overall similarity between an entire source category path, denoted by P, and the sense hierarchy S of the (split) category term. This measure is given by:

$$\text{pathProx}(P, S) = \sum_{x \in P, x \neq \ell} \frac{\text{prox}(x, S)}{|P| - 1} \tag{3}$$

where P is the list of nodes from the source category path, S is a sense hierarchy (one sense from WordNet), and ℓ is the leaf in P. The `pathProx()` returns an average hyperproximity between all the upper categories of a source path and one sense hierarchy from WordNet. Park and Kim use a slightly different function here, which divides the hyperproximities of each upper category by the length of the entire source category path, including the leaf node. This does not lead to a proper average, because no hyperproximity between the leaf node and the sense hierarchy has been calculated.

To illustrate, the `pathProx()` between the source path in Figure 2 and the left sense hierarchy in Figure 3 is equal to 1/9 for our algorithm and equal to 1/15 for the Park and Kim algorithm. This is due to the fact that only the upper category node ‘Appliances’ matches with a node (‘appliance’) in the sense hierarchy. The `prox()` between these nodes is equal to 1/3. Our algorithm divides this number by 3, as it takes the average of the three hyperproximities that it has calculated, which results in a path-proximity of 1/9. Since the Park and Kim algorithm also computes the hyperproximity for the root node ‘Shopping’ and includes the leaf node in the path length count, it divides the total hyperproximity by 5, which results in a path-proximity of 1/15. Note that the path-proximity between the source path and the sense hierarchy on the right of Figure 3 is equal to 0 for both algorithms, since none of the upper categories match with one of the nodes in that sense hierarchy.

3.3.5 Selecting the Correct Sense

Now that we have calculated the path-proximity, using Equation 3, between the source path and each of the possible senses of the source category term, we can determine which one of the senses fits best. This is done by picking the sense hierarchy for which Equation 3 returns the highest value. In our example, which uses the source path depicted in Figure 2 and the sense hierarchies depicted in Figure 3, the sense hierarchy with the highest path-proximity is the left sense hierarchy depicted in Figure 3. Examination of the paths quickly shows that this is the correct decision, as we are looking for a dishwashing machine, rather than a person who washes dishes.

Once the sense hierarchy with the highest path-proximity has been selected, we can obtain the term and its synonyms. This set is called the *extended term set*, which will be used for finding candidate paths in the target taxonomy later on. In the given example, the extended term set consists of all the terms depicted in the light blue (light gray in black and white printing) leaf node depicted on the left of Figure 3. If the original category term was a composite category, we obtain an extended term set for each of the split terms instead. This results in a set consisting of multiple extended term sets, which is called the *extended split term set*.

An important difference between our approach and the Park and Kim approach lies in the way source category terms for which no senses were found in WordNet are handled. Park & Kim suggest that such category terms should not be mapped at all. When our algorithm encounters a similar situation, it returns the extended term set that contains only the original category term. The mapping process then continues as usual. When evaluating the performance of both algorithms, we found that not mapping such category terms by default resulted in a poor performance, particularly with respect to recall. Therefore, the Park & Kim algorithm has been implemented in such a way that it handles these situations in the same way as our algorithm, so that the evaluation gives a more accurate representation of the improvements in performance caused by the other more important changes in our algorithm.

3.4 Candidate Path Identification

The resulting extended (split) term set of the word sense disambiguation process, as described in Section 3.3, is used to identify candidate paths in the target taxonomy. A candidate path is a path in the target taxonomy that is marked by the algorithm as a potential target path to map the current source category to. In order to find the candidate paths, the algorithms compare the terms in the extended (split) term set with the paths in the target taxonomy.

The Park and Kim algorithm starts by comparing the root node of the target taxonomy with the extended term set. If none of the terms in the extended term set is a substring of the currently examined category in the target taxonomy, the algorithm moves one level deeper in the taxonomy and examines the children of the current category. Otherwise, if at least one of the terms in the extended term set is a substring of the currently examined category, that category is marked as a candidate path. In addition, the algorithm no longer considers the children of that path as a potential candidate. The assumption behind this is that if a more general category already matches the term it is more likely to be a better candidate path than a longer and potentially specific category path. However, this does not always hold true for product taxonomies. There are many composite categories in product taxonomies that split the concepts into subcategories later on. For instance, the composite category ‘Music, Movies and Games’ in the Amazon.com [2] product taxonomy has a subcategory called ‘Music’. When one is mapping a category called ‘Music’, it is more appropriate to map it to more specific categories than to more general (composite) categories. Therefore, our algorithm continues to search the entire target taxonomy for candidate paths, even when an ancestor of a path was already marked as a candidate path.

As our algorithm splits the original term if it is a composite category, multiple extended term sets might have to be compared with the category names in the target taxonomy. This means that our algorithm has to perform the matching for each extended term set in its extended split term set. This matching process returns a Boolean value for each extended term set: true if one of the terms is a substring of the currently examined category term, and false if none of the terms is a substring. By aggregating all these Boolean values, we can determine whether the overall match is large enough to consider a category as a candidate path. Ideally, all the concepts – each of which is represented by one extended term set – should have been matched, but this is not always feasible. Rather than imposing a strict match, our algorithm allows for some flexibility by considering categories as candidate paths, if they match with at least half of the extended term sets. Because product taxonomies are loosely defined, this flexibility is needed in order to properly map product taxonomies. For instance, consider the category ‘Computers & Accessories’ in the [31] product taxonomy, which will be split into the concepts ‘Computers’ and ‘Accessories’. Another taxonomy might not have a category that explicitly contains ‘Accessories’, but it might have a category called ‘Computers’. It is obvious that it is preferable to map to this category, but it would not be considered as a candidate path if we demand that it should match each concept from the composite category.

Once all the candidate paths in the target taxonomy have been identified, we need to determine which one of them fits best. In order to calculate the measure of fit, we need to calculate an aggregated similarity score for each candidate path. The aggregated score is composed of two similarity measures, the *co-occurrence* and the *order-consistency*, for the Park and Kim algorithm. Our algorithm also uses these similarity measures, but it also includes an extra measure, called the *parent mapping similarity*. Furthermore, it extends the co-occurrence by splitting terms and using the extended (split) term set of the correct sense. The measures are explained in more detail in the next three sections. It is important to note here that the Park and Kim algorithm only uses the original term or the synonyms of all the senses for the original term, obtained from WordNet, for calculating these similarity measures. The extended term set found by the word sense disambiguation process is no longer used by their algorithm from this point on, while our algorithm continues to use the extended (split) term set, containing only the correct sense of each term.

3.4.1 Co-Occurrence

The co-occurrence is a measure for defining how well each candidate path fits the source category path that one wants to map. It achieves this by using a lexical matching function for each combination of a category from the source and candidate paths. Therefore, it computes the overlap between two category paths, while disregarding the order of nodes in each path. The similarity function is given by:

$$\begin{aligned}\mathrm{coOcc}(P_{\mathrm{src}}, P_{\mathrm{targ}}) = &\left(\sum_{t \in P_{\mathrm{targ}}} \frac{\mathrm{maxSim}(t, P_{\mathrm{src}})}{|P_{\mathrm{targ}}|}\right) \\ &\cdot \left(\sum_{t \in P_{\mathrm{src}}} \frac{\mathrm{maxSim}(t, P_{\mathrm{targ}})}{|P_{\mathrm{src}}|}\right)\end{aligned} \tag{4}$$

where P_{src} and P_{targ} are the list of nodes from the current source path and target path, respectively, and t is a category term.

The `maxSim()` function computes the similarity between a single category name, either from the source or candidate path, and the entire path from the other taxonomy. It compares the single category name with all the nodes in the other path, using a lexical matching function. It will then return the highest similarity score between two category names that it has compared.

The algorithm proposed by Park and Kim uses a simple function to achieve the lexical matching, called `termMatch()`. It checks whether one of the category names is a substring of the other, and returns the length of the shortest string divided by the length of the longest string if this is the case. If this is not the case, the similarity is considered to be zero. While this measure can be computed quickly, it is too strict when used in this way. It cannot deal with the many small lexical variations that appear frequently in product taxonomies.

Furthermore, as the algorithm does not consider the existence of composite categories, the similarity between a composite category and another category is almost always equal to zero, unless the terms happen to appear in the same order in both category names and the punctuation marks are the same. For example, the similarity between 'Home and Garden' and 'Home, Garden & Tools' is equal to zero, because neither string is an exact substring of the other. Nonetheless, this function is useful if one splits the single category name and then compares its separate terms with each category name from the other taxonomy. Therefore, our algorithm uses this adapted version of `termMatch()`.

Furthermore, our algorithm extends the co-occurrence measure by using extended (split) term sets, obtained in the same way as in the candidate path selection process. This ensures that they only contain the synonyms of the correct sense of each term, instead of just all the synonyms. The term t in Equation 4 is replaced by an extended (split) term set of that term. This allows paths with semantically similar terms, which are lexically different, to still achieve a high similarity score.

Our algorithm extends the lexical matching by using an aggregated average similarity score. Instead of only using `termMatch()`, it uses an aggregate of five different similarity measures. Two of the other similarity measures use the Jaccard index [15]. The Jaccard index is used for whole words and for individual characters. Both cases are considered in our algorithm, as they both have their advantages and disadvantages, which will be further explained in more detail later in this section. In addition, our algorithm also uses two Levenshtein distances [18], again using it for whole words and individual characters. Because we want to measure the similarity rather than the distance, a normalized version is used instead. These measures will also be further explained later in this section.

The aggregated similarity score is computed as follows. For each extended term set, the algorithm computes the average similarity score of the five different similarity measures. Then, if there were multiple extended term sets, it will compute the average of their averages to compute the final average of the entire extended split term set.

Like `termMatch()`, the word-based Jaccard index cannot cope with syntactical variations. However, it gives the ratio of matching words, disregarding the length of the words. This can be useful when one has multiple terms with dissimilar lengths. For instance, consider the category 'Binders & Accessories' from [31]. When using a similarity measure where the length of the words matters, like `termMatch()`, it has a bias towards paths that contain the longer words. However, while the term 'Accessories' is longer than 'Binders', it is also general and does not hold much information content. Therefore, it would actually be preferable if a path, which only matches 'Binders', has at least an equal similarity score as a path that matches 'Accessories'. The word-based Jaccard index is a simple and effective measure to achieve this requirement.

In addition to the word-based Jaccard index, the character-based Jaccard index is used as well. Because the order of the characters does not matter, it handles small syntactical variations well. However, it can also introduce errors, because it totally disregards character order. For instance, a word could contain all the characters of another word, but in a different order. In this case the character-based Jaccard index is equal to one, while the words are actually quite dissimilar. However, this drawback is compensated by using an aggregated average similarity score of five different similarity measures.

Character-based Levenshtein distance can be used for strings with lexical variations. Unlike the character-based Jaccard-index, this similarity measure is somewhat stricter, as the order of characters is taken into account. The word-based Levenshtein does not take syntactical variations into account, but it measures how many words the strings have in common and whether the order in which they appear is the same. This is useful when the order of the words can make a difference to the meaning. For example, consider 'Machine Embroidery' and 'Embroidery Machine', referring to machine-made embroidery and a machine which makes embroidery, respectively. Recognizing the difference between nouns and adjectives, which is based on part-of-speech tagging, would be the best way to solve this problem, but it can be fairly complicated and time-consuming to compute. By using the word-based Levenshtein distance, we have a fast method that can also approximately deal with this kind of problem.

3.4.2 Order-Consistency

The co-occurrence is useful for computing the lexical similarity between the source path and a candidate path from the target taxonomy. However, it disregards the order in which these nodes occur in the path. Therefore, we need an additional measure, which takes the order into account. We call this measure the order-consistency, which checks whether the common nodes between the paths appear in the same order.

First, a list of matching nodes, called the *common node list*, between the two paths has to be obtained. The function `common()` adds a node to the list, if it can match the category term of a node, or one of the synonyms of the category term, with a node, or one of its synonyms, from the other path. All the senses from WordNet for the terms are considered here.

The resulting *common node list* is then used by `precedenceRelations()` to create binary node associations. These binary node associations denote a precedence relation between two nodes, which means that the first node occurs before the second node in the hierarchy of the source path. For every element in the *common node list*, pairs of node names from the source path are created.

The `consistent()` function uses the precedence relations to check whether these precedence relations between two nodes also hold true for the candidate path, i.e., whether the two categories in the candidate path occur in the same order as the same two categories in the source path. If a precedence relation still holds true for the candidate path, this function returns the value 1, otherwise it returns 0.

Using the aforementioned functions, the function for the order-consistency is given by:

$$\text{orderCons}(P_{\text{src}}, P_{\text{targ}}) = \sum_{r \in R} \frac{\text{consistent}(r, P_{\text{targ}})}{\binom{\text{length}(C)}{2}} \tag{5}$$

where P_{src} and P_{targ} are the list of nodes from the current source path and target path, respectively, C is $\texttt{common}(P_{\text{src}}, P_{\text{targ}})$, R is $\texttt{precedenceRelations}(C, P_{\text{src}})$, and r is one precedence relation from the source path. The denominator in this function is the number of possible combinations of two nodes, which can be obtained from the *common nodes list*. Therefore, the order-consistency is the average number of precedence relations from the source path that are consistent with the candidate path.

3.4.3 Parent Mapping Similarity

The co-occurrence and the order-consistency both measure the similarity between the source path and a candidate path, computing the degree of category overlap and hierarchical order similarity, respectively. However, we can also exploit our knowledge of how the parent of the source node was mapped in order to find the best candidate path. Because the current source node is closely related to its parent, it is likely that the best candidate path is closely related to the target category to which the parent of the source node was mapped as well.

For example, consider the source path 'Products/Electronics & Computers/Televisions & Video/Brands/Sony' from the [2] product taxonomy. The nodes 'Brands' and 'Sony' could occur in many places in the target taxonomy, because 'Brands' is an abstract concept and 'Sony' is a manufacturer of a large variety of electronic products. This results in multiple candidate paths with the same category name, but with a slightly different hierarchy of nodes preceding them, which makes it difficult to pick the best candidate. However, by applying our knowledge of how 'Products/Electronics & Computers/Televisions & Video' was mapped, we can choose the candidate path that is most closely related to that mapping. This can be easily done by measuring the distance between each candidate path and the category to which the parent was mapped, as both these nodes appear in the target taxonomy. The edge counting technique is used to obtain this distance. However, as we want a degree of similarity, we convert this distance to a normalized measure of similarity. Let $\text{dist}(A, B)$ be the distance between the nodes A and B in the target taxonomy, using the edge counting technique. Then their similarity can be expressed as follows:

$$\text{sim(A, B)} = 1 - \frac{\text{dist}\,(A, B)}{|A| + |B|} \tag{6}$$

3.4.4 Aggregated Similarity Score

Once the various similarity scores, described in the previous sections, have been calculated, we can compute the aggregated similarity score for each candidate path. The Park and Kim algorithm simply takes the arithmetic mean of the co-occurrence and the order-consistency. However, this mean is not robust to outliers. For instance, consider a path for which the co-occurrence was low, but the order-consistency was equal to one. This can happen, as order-consistency only takes the order of common nodes into account. If there are only a few common nodes, there is a considerable chance that their order is consistent. The arithmetic mean would result in an aggregated similarity score of at least 0.5 in this case, which is quite high. This is not intuitive if the paths were actually long and only a few of the nodes matched. In order to better cope with outliers, our algorithm uses the harmonic mean of the co-occurrence, order-consistency, and the parent mapping similarity, instead. The harmonic mean is more appropriate for computing the average of rates, as it has a bias towards low values, mitigating the impact of large outliers. The function for the aggregated similarity score per candidate path is given by:

$$\text{sim}^*(P_{\text{src}}, P_{\text{cnd}}) = \frac{1}{\frac{1}{3}\left(\frac{1}{\text{coOcc}} + \frac{1}{\text{orderCons}} + \frac{1}{\text{parentSim}}\right)} \tag{7}$$

where P_{src} is the list of nodes from the current source path and P_{cnd} is the list of nodes from a candidate path. Because the harmonic mean can only be computed for positive real numbers, a small correction has to be made in order to make it work. If one of the three similarity measures turns out to be zero, it is interpreted as 0.01 in the harmonic mean instead. This low number prevents such candidates from becoming the best candidate path, as the harmonic mean tends to drop sharply when small numbers are involved.

3.5 Best Mapping Selection

Using the overall similarity measures for each candidate path, we can determine which one of the candidates is the best. However, just mapping to the candidate path with the highest similarity, regardless of how high it actually is, is not a good idea. Therefore, a user-configurable similarity threshold is used, which functions as a cut-off. If the overall similarity of the best candidate path is smaller than the threshold, both algorithms will decide not to map to that category. The Park & Kim algorithm maps to nothing in this case. However, our algorithm will look at how the parent of the source category was mapped. If the parent was mapped to a category in the target taxonomy, the current source category will also be mapped to that category. If the parent was not mapped, the current source category will also not be mapped. The rationale behind this decision is that the target taxonomy might not contain the specific category, but it might contain a more general category, which is still better to map to than not mapping at all. For instance, consider the category path 'Online Shopping/Books & Media/Books/Bargain Books', taken from the [31] product taxonomy. If there is no special category for 'Bargain Books' available in the target taxonomy, it would also be acceptable to map this category to the general 'Books' category instead.

4 Evaluation

In this section, we compare the performance of our algorithm with the performance of the Park & Kim algorithm [32] and Anchor-PROMPT [30]. Section 4.1 explains how the evaluation was carried out. It also discusses the used datasets and performance measures. Section 4.2 gives an overview of the performance of the algorithms on these datasets, followed by an analysis of the results.

4.1 Evaluation Design

This section explains how the datasets were obtained and discusses their characteristics. Furthermore, it explains how the datasets were sampled for the evaluation. It also explains how the evaluation was carried out, by discussing how the reference mappings were created and which performance measures were used to compare the algorithms.

4.1.1 Dataset Collection

The algorithms were evaluated for their performance on mapping three real-life datasets. These datasets were obtained using custom-built HTML DOM or RDFa crawlers. The three datasets all have particular characteristics, which gives us the opportunity to assess the performance of the algorithms when coping with different situations.

The largest dataset, containing over 44,000 categories, was obtained from the Open Directory Project (ODP), which is also known as DMOZ [6]. The vast size of this dataset makes it interesting for evaluation purposes, because it shows how well the algorithms scale when mapping large product taxonomies. Furthermore, ODP is actually not a online shop, but a project that attempts to categorize the entire Web with the help of a community of people. Therefore, its categories might differ significantly from those found in online shops, which shows how well the algorithms can cope with (large) syntactical variations. In addition, ODP does not have a proper hierarchical taxonomy either, because there are cross-references between categories and the same categories appear at multiple places in the taxonomy. These cross-references are references between categories, which refer to each other. This would result in an endless loop when constructing the hierarchy structure. Consequently, whenever a category is found that is already in the considered path, it is ignored. While the cross-references have been removed in order to obtain a proper hierarchical taxonomy in the end, the taxonomy still contains many categories with the same name at different places in the taxonomy. This challenges the ability of the algorithms to assess the structural similarity between paths in order to find the best match. The dataset was obtained with a custom-built HTML DOM crawler, which recursively crawled every category found under the root node 'Shopping', up to a maximum depth of five nodes. As previously explained, cross-references were ignored, so that a proper hierarchical taxonomy was obtained in the end.

The second dataset was obtained from [2], the largest online retailer in the United States. This online shop is well-known for its large variety of products, which results in a wide array of concepts represented in the product taxonomy. There are over 2,500 different categories in total, with paths that have a maximum depth of five levels. Therefore, it has both a fairly broad and deep taxonomy structure. Furthermore, as it is one of the largest and best-known online shops, mappings conducted on this dataset give a good indication of how well the algorithms work in practice. The data was obtained using a custom-built HTML DOM crawler, which crawled every category starting with the root node 'Products'.

The last dataset was obtained from O.co, also known as Overstock.com, a large online retailer from the United States. Overstock.com is one of the first major online retailers to annotate all their data using the GoodRelations ontology [13], which has been designed for online shopping. While it has the smallest taxonomy in our datasets, consisting of just over 1,000 categories and with a maximum depth of four levels, it is interesting to be included in the evaluation. It has a comparatively broad and flat taxonomy structure with many composite categories, which makes word sense disambiguation hard. Furthermore, due to its small size, it does not contain all the concepts of the two other taxonomies, which should result in a larger ratio of categories from the other taxonomies that cannot be mapped. The data from this online shop was obtained using a custom-built RDFa crawler, which crawled every category starting with the root node 'Online shopping'.

With the previous three datasets it is possible to perform six different mappings, with each mapping using a different combination of datasets as the source and target taxonomies. In order to evaluate the performance of the algorithms, a human should determine the best mapping for each category in the source taxonomy. This should be done in advance, because retrospective assessment of the performance might easily introduce errors and subjectivity. For example, the ODP dataset contains many categories with the same name, but with a different path. When the algorithm proposes to map to a particular category one might be inclined to agree too quickly with the algorithm, although there might have been another category that was at least as good. It is therefore desirable to create the reference mappings beforehand. Because there is often more than one correct way of mapping a category, determining which category is considered the right category to map to is mostly up to the individual preference of the person creating the mapping. Therefore, in order to reduce the bias in the reference mappings, they were manually made by three individuals, each creating two reference mappings (thus, in total, the required six mappings).

A random sample of five hundred category paths was collected from each of the datasets, because creating the reference mappings for the whole taxonomies would take too much time and effort for our available resources. The samples were randomly taken from the datasets. However, we ensured that for each category node in the sample, its ancestors were also contained in the sample set. This is necessary, as our algorithm also takes into account how the parent path was mapped. Please note that the sample sets are only used for the source taxonomy. The target taxonomy always uses a complete product taxonomy.

4.1.2 Performance Measures

The performance of mapping algorithms is often expressed with a confusion matrix. The confusion matrix shows the number of true positives (TP), false positives (FP), true negatives (TN) and false negatives (FN). The definitions of these classifications in this domain are somewhat different to their general definitions. This is due to the fact that the mapping of product taxonomies is not a binary classification problem, like most classification problems that use a confusion matrix. There are actually many different classes in this case, because every path in the target taxonomy is in fact a separate class. We use the following definitions:

$$\begin{aligned}
\mathrm{TP} &= \#\ \text{correct mappings to a path} \\
\mathrm{FP} &= \#\ \text{incorrect mappings to a path} \\
\mathrm{TN} &= \#\ \text{correct mappings to null} \\
\mathrm{FN} &= \#\ \text{incorrect mappings to null}
\end{aligned}$$

where the positive class P represents the case when a mapping is performed and the negative class N represents the case when no mapping is suggested. Using these classifications, we obtain the following performance measures:

$$\begin{aligned}
\text{sensitivity or recall} &= \frac{\mathrm{TP}}{\mathrm{P}} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}} \\
\text{accuracy} &= \frac{\mathrm{TP} + \mathrm{TN}}{\mathrm{P} + \mathrm{N}} \\
\text{specificity} &= \frac{\mathrm{TN}}{\mathrm{N}} = \frac{\mathrm{TN}}{\mathrm{FP} + \mathrm{TN}} \\
\text{precision} &= \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FP}} \\
F_1\text{-measure} &= 2 \cdot \frac{\text{precision} \ \cdot \ \text{recall}}{\text{precision} \ + \ \text{recall}}
\end{aligned}$$

4.2 Results

This sections presents the results of the mappings, which were performed by our algorithm, the algorithm proposed by Park and Kim, and Anchor-PROMPT. Furthermore, it gives a thorough analysis of the obtained results.

Algorithm	Precision	Recall	F_1-measure	Time
Anchor-PROMPT	28.93%	16.69%	20.75%	0.47 s
Park and Kim algorithm	47.77%	25.19%	32.52%	4.99 s
Our algorithm	38.28%	83.66%	52.31%	20.71 s

Table 1. Comparison of average results per algorithm

4.2.1 Algorithm Mappings

Each algorithm performed a mapping for all the six different combinations of source and target taxonomies, which can be made using the three datasets. The performance measures were obtained by comparing the generated mappings with the reference mappings. In addition, the computation time needed for each mapping was recorded.

Our algorithm and the Park and Kim algorithm have a user-configurable threshold that determines the minimal similarity score for the best candidate path in order to perform a mapping. Because we wanted to determine the optimal threshold value, we ran the algorithms on each mapping using different values for the threshold. We used values ranging between 0 and 1, with a step size of 0.05. Only the best results of these runs were used (for each algorithm and for each mapping) in the reported performances. The average computation times were obtained by taking an average over the computation times of all runs (for each threshold value). This was done to minimize the potential influence of different thresholds on the computation time.

The experiments were run on a computer with an Intel Core 2 Duo P8800 processor at 2.66GHz, with 4GB of memory, running OS X 10.6. The software implementation of the algorithms was done in Java.

4.2.2 Analysis

Table 1 shows the average results of the six mappings per algorithm. Table 2 shows a comparison of computation times per mapping for our algorithm and the Park and Kim algorithm. Tables 3, 4, and 5 show the results with the highest F_1-measure per batch mapping, and the threshold with which this score was achieved, for the Park and Kim algorithm, our algorithm, and Anchor-PROMPT algorithm, respectively.

As shown in Table 1, our algorithm performs better than Anchor-PROMPT and the Park and Kim algorithm on both recall and the F_1-measure. This was especially so for the recall,

Mapping	Park and Kim algorithm	Our algorithm
Amazon → ODP	14.05 s	46.30 s
Amazon → O.co	0.46 s	2.16 s
ODP → Amazon	0.92 s	4.27 s
ODP → O.co	0.42 s	2.04 s
O.co → Amazon	0.79 s	5.04 s
O.co → ODP	13.29 s	64.48 s
Average	4.99 s	20.71 s

Table 2. Computation times per mapping for our algorithm and the Park and Kim algorithm

which represents the ratio of correct mappings of a category and the total amount of categories that should have been mapped, which improved considerably. This measure increased from 16.69% for Anchor-PROMPT and 25.19% for Park and Kim to 83.66% for our algorithm. This also explains the large increase in the F_1-measure for our algorithm, since this value is calculated using both precision and recall. Furthermore, when comparing the results of our previously proposed solution [1], we can conclude that the lexical approach proposed in this paper performs better in terms of precision (38.28% vs. 35.75%), recall (83.66% vs. 45.87%), and F_1-measure (52.31% vs. 39.09%).

Despite the clear improvement in recall, our algorithm actually performed slightly worse on precision than the Park and Kim algorithm. This measure, which represents the ratio of correctly mapped categories to either a particular category or to null, dropped from 47.77% to 38.28%. Analysis of the individual mappings shows that this can be attributed to the increase in false positives. Upon further analysis, we found that the Park and Kim algorithm performs well for categories that should be mapped to null, but this is offset by a relatively poor performance on correctly mapping to a particular category, i.e., categories that should be mapped but were not mapped. Therefore, the overall performance of the Park and Kim algorithm, the F_1-measure, is considerably lower than that of our algorithm. We argue that recall is more important within the domain of product taxonomy mapping, as the task of determining incorrect mappings is easier than the task of finding correct mappings (because of the large search space). Therefore, the slight decrease in precision for our algorithm is well compensated by its higher recall.

Our algorithm performs better on precision, recall, and the F_1-measure than Anchor-PROMPT, which maps more conservatively due to the fact that it is geared towards ontology mapping in general. The Park and Kim algorithm also outperforms this algorithm on the three measures. From this, we can conclude that algorithms that are specifically tailored to mapping product taxonomies, such as our algorithm and that of Park and Kim, have a significant advantage over more general ontology mapping algorithms within this domain.

Another interesting observation is the comparatively long computation times for both the Park and Kim algorithm and our algorithm. Anchor-PROMPT appears to sacrifice precision and recall for a fast computation time, whereas the other algorithms focus more on better precision and recall. The increase in the average computation time of 4.99 seconds for Park and Kim to 20.71 seconds for our algorithm can be largely attributed to the splitting of category names by our algorithm. Our algorithm has to calculate many of the functions in the algorithm for each individual word in a category name, rather than calculating it once for the entire category name, like the Park and Kim algorithm does. Despite the longer computation times for these algorithms, they still scale well enough in order to be effectively used in practice. For instance, the complete dataset from Amazon.com has 2,500 categories, which means that it would probably take about 25 seconds (5×4.99s) on average for the Park and Kim algorithm, and 100 seconds (5×20.71s) on average for our algorithm, to map all the categories from Amazon.com to another taxonomy. Furthermore, as Table 2 shows, the mappings to ODP increase the average computation time considerably for both algorithms. Because product taxonomies from Web shops are unlikely to contain as many categories as ODP, the algorithms are probably faster on average when used exclusively on product taxonomies obtained from real Web shops (e.g., Amazon.com and Overstock.com),

Mapping	Precision	Accuracy	Specificity	Recall	F_1-measure	Threshold
Amazon → ODP	35.77%	34.00%	57.89%	16.84%	22.90%	0.05
Amazon → O.co	60.16%	47.20%	76.78%	25.61%	35.92%	0.00
ODP → Amazon	37.06%	41.48%	51.94%	30.29%	33.33%	0.00
ODP → O.co	36.76%	35.87%	48.68%	25.09%	29.82%	0.10
O.co → Amazon	61.14%	36.20%	52.11%	29.89%	40.15%	0.00
O.co → ODP	55.71%	36.60%	62.87%	23.42%	32.98%	0.50
Average	47.77%	38.56%	58.38%	25.19%	32.52%	0.11

Table 3. Best results for Park and Kim algorithm

Mapping	Precision	Accuracy	Specificity	Recall	F_1-measure	Threshold
Amazon → ODP	29.60%	29.60%	4.85%	77.65%	42.86%	0.25
Amazon → O.co	53.13%	58.40%	26.69%	94.44%	68.00%	0.15
ODP → Amazon	35.36%	45.89%	27.94%	84.28%	49.81%	0.20
ODP → O.co	42.29%	50.30%	29.77%	83.68%	56.18%	0.15
O.co → Amazon	37.15%	40.40%	13.78%	84.57%	51.62%	0.15
O.co → ODP	32.13%	36.00%	14.33%	77.33%	45.39%	0.20
Average	38.28%	43.43%	19.56%	83.66%	52.31%	0.18

Table 4. Best results for our algorithm

Mapping	Precision	Accuracy	Specificity	Recall	F_1-measure	Threshold
Amazon → ODP	2.83 %	14.40%	12.71%	42.86%	5.30%	-
Amazon → O.co	11.46%	29.40%	23.25%	77.19%	19.95%	-
ODP → Amazon	6.32 %	23.87%	20.89%	54.55%	11.32%	-
ODP → O.co	7.29 %	20.45%	16.33%	54.71%	12.86%	-
O.co → Amazon	17.62%	28.00%	16.02%	84.09%	29.13%	-
O.co → ODP	3.47 %	18.80%	17.02%	46.67%	6.45%	-
Average	8.17 %	22.49%	17.70%	60.01%	14.17%	-

Table 5. Best results for Anchor-PROMPT

which is the most practical use for these algorithms.

Further analysis of the results per mapping, shown in Tables 3 and 4, suggests that the optimal threshold is close to 0.11 for Park and Kim and approximately 0.18 for our algorithm. This can be explained by the different method employed by our algorithm to perform the lexical matching between category names. The Park and Kim algorithm uses a strict lexical matching function, in which one category string has to appear entirely in another category string. In addition, our algorithm employs approximate string matching functions, such as the Levenshtein distance [18] and the Jaccard index [15] that operates on sets with the individual characters of a category string as the elements. By using these measures as well, our algorithm is better able to cope with syntactical variations within category strings, and the lexical similarity will increase overall. To counter the side-effect of higher similarity scores in general by using approximate string matching techniques, the threshold needs to be adjusted accordingly.

From Tables 3 and 4 we can also conclude that the recall of our algorithm for each mapping has improved considerably, compared to the Park and Kim algorithm. Particularly in the mappings where Overstock.com is used as the target taxonomy, we found a tremendous increase in recall. The recall for the mapping from Amazon.com to Overstock.com increased from 25.61% to 94.44%. In addition, the recall for the mapping from ODP to Overstock.com changed from 25.09% to 83.68%. Because Overstock.com contains many composite categories, the Park and Kim algorithm is often not able to map these. Furthermore, mapping to the category to which the parent is mapped helps in situations where no suitable candidate can be found. Because Overstock.com is the smallest taxonomy, it does not always contain the same detailed concepts as the other taxonomies, but it does have the more general concepts to which these concepts can be mapped.

We also notice that the precision is approximately the same for the mapping from ODP to Amazon.com. In this case, the algorithms have difficulties in identifying candidate paths in the target taxonomy, because of the syntactical variations (compared to product taxonomies from real online shops) present in the ODP dataset. However, for ODP to Overstock.com, we find that our algorithm outperforms the Park and Kim algorithm. The precision is increased from 36.76% to 42.29%. In this case, our algorithm suffers less from the previously mentioned difficulties because of the approximate string matching functions used. This suggests that for some data sets, the precision is also influenced by the approximate string matching techniques employed.

5 Conclusion

This paper proposes an algorithm that is suitable for automated product taxonomy mapping in an e-commerce environment. The main focus of the algorithm is to take into account the domain-specific characteristics of product taxonomies, like the existence of composite categories and the syntactical variations in category names. Our algorithm has been shown to perform better than the other approaches when mapping product taxonomies. It manages to significantly increase the recall of the mappings, which is more than three times higher than the recall of the other algorithms. Furthermore, the F_1-measures increase from 14.17% for Anchor-PROMPT and 32.52% for the Park and Kim algorithm, to 52.31% for our algorithm. Despite these improvements, the precision decreases from 47.77% for the Park and Kim algorithm to 38.28% for our algorithm. This is explained by the increase of false positives for our algorithm, i.e., categories that have been incorrectly mapped. Despite the slight decrease in precision, we believe that our algorithm, having an average recall above 80%, contributes significantly to the area of product taxonomy mapping. We argue that in the context of e-commerce and product taxonomies, recall is more important than precision. The main reason for this is that the task of identifying whether a mapping is incorrect or not is easier than finding a correct mapping, as the search space is relatively large in this context.

For future work, we would like to employ a more advanced word sense disambiguation technique, such as the one proposed by [17]. It would also make sense to consider using word category disambiguation, also known as part-of-speech tagging, for the words found in a category name. Often the meaning changes when the part-of-speech is different. For example, 'Machine Embroidery' and 'Embroidery Machines', referring to machine-made embroidery and a machine that makes embroidery, respectively. By differentiating between adjectives and nouns, it is possible to identify these two meanings in product categories.

Acknowledgment

Damir Vandic is supported by an NWO Mozaiek scholarship for his project entitled *Semantic Web Enhanced Product Search (SWEPS)*, project 017.007.142.

References

1. Steven S Aanen, Damir Vandic, and Flavius Frasincar. Automated Product Taxonomy Mapping in an E-commerce Environment. *Expert Systems with Applications*, 42(3):1298–1313, 2015.
2. Amazon.com. US' Largest Online Retailer. `http://www.amazon.com`, 2015.
3. David Aumueller, Hong-Hai Do, Sabine Massmann, and Erhard Rahm. Schema and Ontology Matching with COMA++. In *ACM SIGMOD International Conference on Management of Data 2005 (SIGMOD 2005)*, pages 906–908. ACM, 2005.
4. Silvana Castano, Alfio Ferrara, and Stefano 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.
5. Silvana Castano, Alfio Ferrara, Stefano Montanelli, and Daniele Zucchelli. Helios: A General Framework for Ontology-Based Knowledge Sharing and Evolution in P2P Systems. *14th International Workshop on Database and Expert Systems Applications (DEXA 2003)*, 2003.
6. DMOZ. Open Directory Project. `http://www.dmoz.org/`, 2015.
7. Hong-Hai Do and Erhard Rahm. COMA: A System for Flexible Combination of Schema Matching Approaches. In *28th International Conference on Very Large Data Bases (VLDB 2002)*, pages 610–621. VLDB Endowment, 2002.
8. Mark Ehrig and Steffen Staab. QOM - Quick Ontology Mapping. In *International Semantic Web Conference 2004 (ISWC 2004)*, volume LNCS-3298, pages 683–697. Springer, 2004.
9. John H. Gennari, Mark A. Musen, Ray W. Fergerson, William E. Grosso, Monica Crubézy, Henrik Eriksson, Natalya F. Noy, and Samson W. Tu. The Evolution of Protégé: An Environment for Knowledge-based Systems Development. *International Journal of Human-Computer Studies*, 58(1):89–123, 2003.
10. Fausto Giunchiglia, Pavo Shvaiko, and Mikalai Yatskevich. Semantic Schema Matching. *On the Move to Meaningful Internet Systems 2005: CoopIS, DOA, and ODBASE*, pages 347–365, 2005.
11. Thomas R. Gruber. A Translation Approach to Portable Ontology Specifications. *Knowledge Acquisition*, 5:199–199, 1993.
12. Bin He and Kevin Chen-Chuan Chang. Automatic Complex Schema Matching across Web Query Interfaces: A Correlation Mining Approach. *ACM Transactions on Database Systems*, 31(1):1–45, 2006.
13. Martin 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 LNCS-5268, pages 329–346. Springer, 2008.
14. John B. Horrigan. Online Shopping. *Pew Internet & American Life Project Report*, 36, 2008.
15. Paul Jaccard. The Distribution of the Flora in the Alpine Zone. *New Phytologist*, 11(2):37–50, 1912.
16. Yannis Kalfoglou and Marco Schorlemmer. Ontology Mapping: The State of the Art. *The Knowledge Engineering Review*, 18(01):1–31, 2003.
17. Michael Lesk. Automatic Sense Disambiguation using Machine Readable Dictionaries: How to tell a Pine Cone from an Ice Cream Cone. In *5th Annual ACM SIGDOC International Conference on Systems Documentation (SIGDOC 1986)*, pages 24–26. ACM, 1986.
18. Vladimir I. Levenshtein. Binary Codes Capable of Correcting Deletions, Insertions, and Reversals. *Soviet Physics Doklady*, 10(8):707–710, 1966.
19. John Li. LOM: A Lexicon-based Ontology Mapping Tool. In *5th Workshop Performance Metrics*

for Intelligent Systems (PerMIS 2004), 2004.

20. Lei Li, Wenxing Hong, and Tao Li. Taxonomy-Oriented Recommendation Towards Recommendation with Stage. In *Web Technologies and Applications*, pages 219–230. Springer, 2012.
21. Jayant Madhavan, Philip A. Bernstein, and Erhard Rahm. Generic Schema Matching with Cupid. In *27th International Conference on Very Large Data Bases (VLDB 2001)*, pages 49–58. Morgan Kaufmann Publishers Inc., 2001.
22. Bernardo Magnini, Manuela Speranza, and Christian Girardi. A Semantic-Based Approach to Interoperability of Classification Hierarchies: Evaluation of Linguistic Techniques. In *20th International Conference on Computational Linguistics (COLING 2004)*, pages 1133–es. Association for Computational Linguistics, 2004.
23. Deborah L. McGuinness, Richard Fikes, James Rice, and Steve Wilder. The Chimaera Ontology Environment. In *17th National Conference on Artificial Intelligence (AAAI 2000)*, pages 1123–1124, 2000.
24. Sergey Melnik, Hector Garcia-Molina, and Erhard Rahm. Similarity Flooding: A Versatile Graph Matching Algorithm and its Application to Schema Matching. In *18th International Conference on Data Engineering (ICDE 2002)*, pages 117–128. IEEE Computer Society, 2002.
25. George A. Miller. WordNet: A Lexical Database for English. *Communications of the ACM*, 38(11):39–41, 1995.
26. Lennart Nederstigt, Damir Vandic, and Flavius Frasincar. An Automated Approach to Product Taxonomy Mapping in E-Commerce. In *1st International Symposium on Management Intelligent Systems (IS-MiS 2012)*, volume 171 of *Advances in Intelligent Systems and Computing*, pages 111–120. Springer, 2012.
27. Hien T Nguyen and Tru H Cao. Named Entity Disambiguation: a Hybrid Approach. *International Journal of Computational Intelligence Systems*, 5(6):1052–1067, 2012.
28. Ian Niles and Adam Pease. Towards a Standard Upper Ontology. In *2nd International Conference on Formal Ontology in Information Systems (FOIS 2001)*, pages 2–9. ACM, 2001.
29. Ian Niles and Allan Terry. The MILO: A General-purpose, Mid-level Ontology. In *2nd International Conference on Information and Knowledge Engineering (IKE 2004)*, pages 15–19, 2004.
30. Natalya F. Noy and Mark A. Musen. The PROMPT Suite: Interactive Tools for Ontology Merging and Mapping. *International Journal of Human-Computer Studies*, 59(6):983–1024, 2003.
31. Overstock.com. Web Store. `http://www.o.co`, 2015.
32. Sangun Park and Wooju Kim. Ontology Mapping between Heterogeneous Product Taxonomies in an Electronic Commerce Environment. *International Journal of Electronic Commerce*, 12(2):69–87, 2007.
33. Erhard Rahm and Philip A. Bernstein. A Survey of Approaches to Automatic Schema Matching. *The VLDB Journal*, 10(4):334–350, 2001.
34. Shopping.com. Online Shopping Comparison Website. `http://www.shopping.com`, 2015.
35. Pavel Shvaiko and Jerome Euzenat. A Survey of Schema-Based Matching Approaches. *Journal on Data Semantics IV*, 3730:146–171, 2005.
36. Guo-Qing Zhang, Guo-Qiang Zhang, Qing-Feng Yang, Su-Qi Cheng, and Tao Zhou. Evolution of the Internet and its Cores. *New Journal of Physics*, 10(12):123027, 2008.
37. Hongwei Zhu, Stuart Madnick, and Michael Siegel. Enabling Global Price Comparison through Semantic Integration of Web Data. *International Journal of Electronic Business*, 6(4):319–341, 2008.
38. Cai-Nicolas Ziegler, Georg Lausen, and Lars Schmidt-Thieme. Taxonomy-Driven Computation of Product Recommendations. In *13th International Conference on Information and Knowledge Management (CIKM 2004)*, pages 406–415. ACM, 2004.