Journal of Web Engineering, Vol. 13, No. 3&4 (2014) 277–301
© Rinton Press

IMPROVING SEARCH AND EXPLORATION IN TAG SPACES USING AUTOMATED TAG CLUSTERING

JONI RADELAAR, AART-JAN BOOR, DAMIR VANDIC
JAN-WILLEM VAN DAM, and FLAVIUS FASINCAR
Econometric Institute, Erasmus University Rotterdam
Burgemeester Oudlaan 51, PO Box 1738, NL-3000 DR
Rotterdam, the Netherlands
joni@radelaar.nl, aartjan.boor@gmail.com, vandic@ese.eur.nl, jwvdam@gmail.com, frasincar@ese.eur.nl

Received July 15, 2013
Revised April 3, 2014

In recent years we have experienced an increase in the usage of tags to describe resources. However, the free nature of tagging presents some challenges regarding the search and exploration of tag spaces. In order to deal with these challenges we propose the Semantic Tag Clustering Search (STCS) framework. The framework first groups syntactic variations using several measures based on the Levenshtein distance and the cosine similarity based on tag co-occurrences. We find that a measure that combines the newly introduced variable cost Levenshtein similarity measure with the cosine similarity significantly outperforms the other methods we evaluated in terms of precision. After grouping syntactic variations, the framework clusters semantically related tags using the cosine similarity based on tag co-occurrences. We compare the STCS framework to a state-of-the-art clustering technique and find that the STCS framework performs significantly better in terms of precision. For the evaluation we used a large data set gathered from Flickr, which contains all the pictures uploaded in the year 2009.

Keywords: Tagging, syntactic clustering, semantic clustering, tag disambiguation

1 Introduction

On today's World Wide Web, it is becoming increasingly popular to use tags to describe resources. Tags are freely chosen keywords that can be used to annotate resources, such as videos, photos, or Web pages. The advantage of using tags is that, because there are no restrictions on the tags that can be used, they provide a flexible way of describing resources [21]. The disadvantage of using tags is that, because of the unstructured nature of tagging, there are some problems associated with retrieving resources using tag-based searches [10, 20].

One example of a problem with tag-based searches is that it is possible that different tags have the same or closely related meanings. This can be caused by syntactic variations, but it can also be the result of the use of synonyms [9]. Examples of syntactic variations are misspellings or the use of the grammatical number form of a specific word. Golder and Huberman [13] describe the basic level variation problem, which is an issue in tag-based searches. This problem refers to the fact that different users may employ different levels of granularity to describe a resource. For example, one user might tag a picture of a dog as "animal" (not very specific), while another user would use "Dalmatian" (very specific). The use of homonyms is another problem when tagging a resource. A homonym is a word with multiple meanings. For example, “Turkey” refers to both the country and the animal.

The problems described above can lead to difficulties when searching for resources using tags. For example, if a user is looking for a picture using “dog” as a keyword, he or she would most likely also be interested in pictures tagged with “dogs” (syntactic variation), “Dalmatian” (more specific, semantically related term), “doggy” (synonym), and “dough” (misspelling of dog).

One way to tackle these problems is to create clusters of syntactically and semantically related tags. Creating syntactic clusters involves the grouping of tags that are syntactic variations of each other. An example of such a cluster is {“parijs”, “paris”}. Creating semantically related clusters involves grouping tags that are semantically related. An example of such a cluster would be {“paris”, “notredame”}. Tags with multiple meanings can then be identified by checking whether a tag occurs in multiple semantic clusters. If a tag occurs in multiple clusters it most likely also has multiple meanings, e.g., “bow” can refer to both the weapon and the front of a ship.

Search algorithms can utilize these syntactic and semantic clusters in order to improve the quality of search query results. For example, when a user enters a tag as a search query, the search algorithm could also include in the result pictures that are tagged with a syntactic or semantic variation of the entered query, and therefore improving recall. Also, if a tag is ambiguous, the user can be offered the choice to select between the different tag meanings (defined by the semantic clusters) and thus improve the precision of the query results.

Several clustering techniques for tags have been previously proposed [7, 28, 30, 35]. However, the evaluation of these techniques is done using relatively small datasets with at most several hundreds of thousands resources. In this article we describe the Semantic Tag Clustering Search (STCS) framework that supports several clustering techniques. We evaluate different syntactic and semantic clustering techniques using a dataset that contains more resources than previously used, i.e., tens of millions. The data for our experiments is retrieved from Flickr.com [1], a very popular photo sharing website. To construct a large data set we use all images uploaded on Flickr in 2009.

We also develop a Web application that is based on the proposed framework and allows users to search for pictures using tags and presents the results of the search using several clustering techniques. For each query the user can experiment with several clustering techniques and choose the one that produces the most adequate results. This gives us insight into which clustering methods produce the best results for different real-world end-users. Statistical tests are used to analyze if our search engine performs better than existing alternatives.

The contributions of this paper to searching and browsing tag-spaces are four-fold: it proposes a new syntactic clustering algorithm, adapts an existing semantic algorithm with two new heuristics, evaluates the syntactic and semantic clustering algorithm on a larger data set than previously used in related work, and implements a Web application with different methods for searching and browsing tag spaces.

The work presented in this paper consolidates our previous results on syntactic and semantic clustering for searching tag spaces [25, 29, 33, 31, 32, 34] and proposes novel techniques for clustering in our framework, such as advanced data cleaning operations and a variable cost Levenshtein distance. In [29, 33] we laid the foundations for the STCS framework, which included a syntactic and semantic clustering step. In [25, 32] we investigated the use of a graph clustering algorithm that is based on community detection in graphs. We focused on the scalability of our framework in [31, 34] and provided solutions for the found bottlenecks. Different from previous publications, in this paper we perform an in-depth sensitivity analysis of the framework parameters and provide the details of the Web application that implements the proposed framework. We also experiment with a novel variable-cost Levenshtein similarity measure that improves the syntactic clustering process. Furthermore, for the evaluation, we use larger data sets than previously reported in the literature.

2 Related Work

In this section we discuss the work related to the research done in this paper. We have divide the discussion over three subsections on tag clustering, similarity measures, and cluster evaluation, as these are the most important topics that we address in our framework.

2.1 Tag Clustering

Echarte et al. [11] discuss the influence of syntactic variations on the quality of folksonomies. To improve the quality of folksonomies they propose to use pattern matching techniques, such as the Levenshtein [17] and Hamming [14] distances, to identify syntactic variations. Using data from CiteULike the authors evaluate the performance of the Levenshtein and Hamming distances. The data set used to evaluate the performance of the two measures for identifying syntactical variations contained the 10,000 most popular tags and 1,577,198 annotations. One annotation consists of a user assigning one tag to a resource. The outcome of this study is that overall the Levenshtein distance performs better than the Hamming distance. However, both techniques do not perform well with tags shorter than 4 characters. In a later paper, Echarte et al. [12] introduce a new fuzzy similarity measure for grouping syntactic variations of tags. They compare this new measure with the Levenshtein and Hamming distances and find that it performs significantly better in identifying syntactic variations.

Specia and Motta [28] introduce a method for building clusters of semantically related tags using a non-hierarchical clustering technique based on tag co-occurrence. They also explore the relationships between pairs of tags within a cluster. The cosine similarity among tags given by their co-occurrence vectors is used as a similarity measure in the clustering process. Before creating the clusters, the authors merge morphologically similar tags using the normalized Levenshtein similarity measure. Data used for the experiment is gathered from Flickr and Delicious. The Flickr data set contained 49,087 distinct resources and 17,956 distinct tags, and the Delicious data set contained 14,211 distinct resources and 11,960 distinct tags. The results have been evaluated manually and the authors found that the clustering approach results in meaningful groups of tags corresponding to concepts in ontologies.

Begelman et al. [7] propose to build a directed graph where tags are represented by vertices. An edge between two vertices indicates a (strong) relation between the two tags represented by the vertices. The weight of the edge is based on the co-occurrence of the two tags the edge connects. The authors employ an algorithm that recursively uses spectral bisection to split the graph in two clusters. This split is then evaluated using the modularity function, which was introduced by Newman and Girvan [23], and provides a measure of the quality of a particular division of a network. The clustering algorithm has been applied to a dataset containing 200,000 resources and 30,000 tags.

Yueng et al. [35] also use a graph-based clustering algorithm. Like Begelman, the authors use the modularity function to evaluate the quality of a division. However, in addition they also investigate the effect of various network representations of tags and resources on the resulting clusters of semantically related tags. These different representations include networks based on co-occurrence of tags, and context of tags using cosine similarity. For their research, the authors used a small data set gathered from Delicious, by manually selecting 20 tags that are used to represent two or more concepts and complementing these tags by randomly selecting 30 tags from the 100 most popular tags. For each tag they then gathered about 500 images that are annotated with that tag. After applying the clustering algorithm to the used tag representations, the resulting clusters are evaluated manually. The conclusion of their experiment is that networks based on tag context similarity capture the most concepts. These networks use the cosine similarity to compare the context in which two tags are used, as reflected by the co-occurrence vectors of the tags. In addition the results produced by the clustering algorithm are compared with WordNet synset relations. The results of this comparison is that the clustering algorithm detects more semantically related tags than are derived using WordNet [22] synsets.

Schmitz [27] proposes a subsumption-based model to derive a hierarchy of semantically related Flickr tags. Schmitz uses an adaptation of the simple statistical subsumption model described by Sanderson and Croft [26]. In the model proposed by Schmitz the statistical threshold for the subsumption of a tag is adjusted dynamically and filters are added to control for the highly idiosyncratic vocabulary of Flickr tags. This model is then used to derive trees of semantically related tags. For the experiment Schmitz gathered an initial dataset of approximately 9 million images, 200,000 tags, and 8 million tag pairs. However, the exact amount of data used to derive the trees in unknown, since the algorithm filters the data during the clustering process to reduce memory usage. The resulting trees were evaluated manually. The result of this evaluation is that the adapted model works better than the original model described by Sanderson and Croft [26].

All of the above mentioned methods suffer from two main issues, which are addressed by our approach. First, none of the methods deals with the problem of uneven cluster sizes. In tagging spaces, some clusters can be very large while others can be rather small. This can result in poor performance in the cluster merging process. Second, none of the methods have a robust syntactic similarity measure that gives different weights to various mistake types made by users. This is another important aspect in tagging spaces as tags are typed by everyday users and some type of mistakes occur more often than others.

2.2 Similarity Measures

There are many measures that can be used to determine the similarity between two tags. Cattuto et al. [8] evaluate a few similarity measures using a Delicious dataset containing the 10,000 most popular tags. They perform this evaluation by comparing the relations established by using the different similarity measures with WordNet synsets. The experiment shows that the cosine similarity is the best similarity measure for detecting synonyms, while FolkRank and co-occurrence appear to be most useful for detecting various other semantic relations, such as type-of and multi-world relations.

Markines et al. [19] evaluate several similarity measures, including matching similarity, overlap similarity, Jaccard similarity, Dice coefficient, cosine similarity, and mutual information, using a more systematic approach. They evaluate the performance of these measures by generating several two-dimensional views on the tripartite data gathered from BibSonomy.org. Their initial dataset contained 128,500 resources, 1,921 users, and 58,753 tags. However, for the evaluation, the initial dataset is reduced to the tags that occur in WordNet (17,041 tags) and the 2000 most popular resources. The two-dimensional view is generated by using several aggregation methods. An important difference between BibSonomy.org and Flickr is that in Flickr only one user can tag a resource. Therefore, for Flickr data only projection aggregation is useful. The result of projection aggregation can be seen as a matrix with binary elements $w_{rt} \in \{0,\ 1\}$ where rows correspond to resources and columns corresponds to tags. Given a resource and a tag, a 0 in a matrix cell means that no user associated that resource with that tag, whereas a 1 means that at least one user has performed the indicated association. All similarity measures can then be derived directly from this information.

The evaluation of the various tag-tag similarity measures in the previous study is done using Kendall τ correlations between the similarity vectors generated by the various measures and a reference similarity, given by the Jiang-Conrath distance [15] between terms in WordNet. The outcome of this evaluation is that mutual information is the best similarity measure when using projection aggregation. The remaining similarity measures have the same performance when compared to each other. However, because mutual information is a computationally-intensive measure, its use is unfeasible for large data sets.

We use the algorithm of Specia and Motta [28] as a starting point and improve upon it by adapting their heuristics for merging clusters. We also build upon the idea of using a fuzzy clustering technique to improve the results of syntactic clustering, as proposed by Echarte et al. [12]. For our experiment we use a large dataset that contains more resources than previously used in similar research.

2.3 Evaluation

In the literature different measures have been used to evaluate clusters. Larsen and Aone [16] describe the precision measure. Average precision is defined as follows:

$$\mathrm{AP}(\Omega, C) = \frac{1}{|\Omega|} \sum_{w_k \in \Omega} \frac{\max\limits_{c_j \in C} |\omega_k \cap c_j|}{|w_k|} \tag{1}$$

where $\Omega = \{\omega_1, \omega_2, ..., \omega_k\}$ is the set of computed clusters and $C = \{c_1, c_2, ..., c_j\}$ is the set of desired clusters. We interpret ω_k as the set of tags in ω_k and c_j as the set of tags in c_j.

Manning et al. [18] present the purity measure to evaluate clusters. They define purity as follows:

$$\mathrm{purity}(\Omega, C) = \frac{1}{N} \sum_{w_k \in \Omega} \max_{c_j \in C} |\omega_k \cap c_j| \tag{2}$$

where Ω and C are the same as in the previous equation and N is the total number of tags in Ω (including duplicates). We use these two measures to evaluate the different clustering techniques.

3 Framework Design

To address the issues mentioned in the introduction, we propose to use the Semantic Tag Clustering Search (STCS) framework. The framework consists of two distinct layers: removing syntactic variations and finding semantically related tags. In the first layer we address syntactic variations of tags, like misspellings and morphological variations. These variations are eliminated by creating clusters of tags that are syntactic variations of each other and assigning a label, i.e., a unique tag, to each cluster. The second layer of the framework addresses the problem of identifying semantically related tags. In this section we first define the problem formally. Then, we give the used similarity measures and present the STCS framework.

3.1 Problem Definition

The input for the framework is the data set that is defined as a tuple $D = \{U, T, P, r\}$, where U, T, and P are the finite sets of users, tags, and pictures, respectively. The ternary relationship $r \subseteq U \times T \times P$ specifies the initial annotations of the users. The problem definition for the two distinct layers of the STCS framework is discussed below in separate sections.

3.1.1 Removing Syntactic Variations

To make the detection of semantically related tags effective, it is useful to first remove syntactic variations of tags from the data set. Syntactic variations are usually misspellings of words, but may include translations of tags in other languages or morphological variations. An example of a syntactic variation, in which one of the tags is the plural form of the other tag is “nutcracker” and “nutcrackers”.

To remove these syntactic variations we create a set of tag sets $T' \subset \mathcal{P}(T)$. Each element of the set T' is a cluster that contains all tags that are syntactic variations of each other under the condition that each tag can only appear in one cluster. To determine the label of a cluster, we define m' which is the bijective function that indicates a label for each $x \in T'$, $m' : T' \rightarrow L$. For each $l \in L$ and some $x \in T'$, $l \in x$ holds, thus, l is one of the tags from cluster x that is selected as the label.

3.1.2 Finding Semantically Related Tags

The second layer of the framework aims at creating clusters of tags that are semantically related. For this purpose, we define a set T'' in which each element is a cluster of elements $l \in L$. By only clustering tags that are labels of syntactic clusters we disregard the syntactic variations in the semantic clusterings. An example of a semantic cluster is {“necklace”, “bracelet”, “earrings”, “beads”}. A tag can be part of multiple clusters, each with a different meaning. An example that presents the different types of clusters can be seen in Fig. 1.

3.2 Similarity Measures

In this section we discuss the similarity measures used in the framework, i.e., the Levenshtein distance, variable cost Levenshtein distance, and cosine similarity.

3.2.1 Levenshtein Distance Measure

The Levenshtein distance [17] measures the amount of typographic difference between two strings, which is also called the edit distance. The Levenshtein distance between two strings

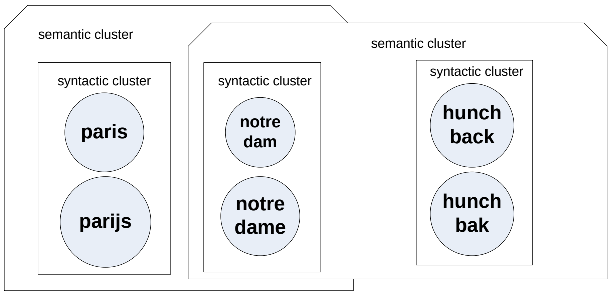

Fig. 1. Example of syntactic and semantic clusters

is defined by the minimum number of edit operations needed to transform one string into the other. An operation is defined as an insertion, deletion, or substitution of a single character.

For example, the Levenshtein distance between ‘pair’ and ‘stairs’ is 3, since the following three edit operations change one string into the other, and it is not possible to achieve this result with fewer than three edits:

1. pair $\rightarrow$ spair (insert of ‘s’ at the beginning)
2. spair $\rightarrow$ stair (substitution of ‘p’ for ‘t’)
3. stair $\rightarrow$ stairs (insert ‘s’ at the end).

We denote this distance by alv_{ij}, which is the absolute Levenshtein distance between tag i and j because it does not considers the length of the strings. One can argue that a Levenshtein distance of three is more significant if the edit distance between two strings of 5 characters than if both strings consist of 12 characters.

The framework needs to be able to deal with varying tag lengths as shown above, that is why we use the normalized Levenshtein similarity, which is a measure that is relative to the tag length. The normalized Levenshtein similarity between tag i and j, denoted by lv_{ij}, is defined as

$$lv_{ij} = 1 - \frac{alv_{ij}}{\max(length(t_i), length(t_j))} \tag{3}$$

3.2.2 Variable Cost Levenshtein Distance Measure

The traditional Levenshtein distance assumes that each insertion, substitution, or deletion of a character has the same contribution with regards to the overall cost, i.e., each operation adds to the distance a fixed amount of cost. This approach produces limited results when aiming to identify syntactic variations. The addition of an ‘s’ at the end of a word, to create the plural form of the word, contributes less to the distance between two words than replacing an ‘a’ with a ‘u’ somewhere in the middle of a word. Inspired by the fuzzy similarity measure proposed by Echarte et al. [12], we propose the Variable Cost Levenshtein distance, denoted by $avclv_{ij}$, which addresses this previously identified problem. Unlike Echarte et al. we use a clustering algorithm instead of a fuzzy finite state automaton for computing syntactical variations. We also use different costs for different operations and combine the variable cost Levenshtein with the cosine similarity. We define the variable cost based on the following heuristic rules:

Parameter Rule 1	**Parameter Rule 2**	**Parameter Rule 3**	**Precision**	**Recall**	**F-measure**
0.5	0.5	100	0.88	0.88	0.88
0.5	0.5	50	0.88	0.88	0.88
0.5	0.5	10	0.88	0.88	0.88
0.5	0.5	5	0.88	0.88	0.88
0.5	0.5	3	0.87	0.88	0.87
0.5	0.5	2	0.82	0.88	0.85
0.5	0.5	1	0.72	0.88	0.79
0.5	0.5	0.5	0.50	0.88	0.64
0.5	1.0	100	0.87	0.88	0.87
0.5	0.75	100	0.87	0.88	0.87
0.5	0.5	100	0.88	0.88	0.88
0.5	0.25	100	0.87	0.88	0.87
0.5	0	100	0.87	0.88	0.87
1	0.5	100	0.87	0.73	0.79
0.75	0.5	100	0.87	0.86	0.86
0.5	0.5	100	0.88	0.88	0.88
0.25	0.5	100	0.87	0.89	0.88
0	0.5	100	0.88	0.78	0.83

Fig. 2. Performance of the Variable Cost Levenshtein Distance

1. The addition of a ‘s’ at the end of a word costs 0.5.
2. An operation involving a non-alphanumerical character costs 0.5.
3. An operation involving two numerical characters costs 100.
4. Any other operation costs 1.

We have tested these rules on a sample dataset with 1100 tags that contain syntactical variations and found this combination of rules and costs to provide good precision and recall results on our sample without making the measure too complex and thus slow. Because we wanted to reduce the cost of the operations considered by rule 1 and 2, for these rules we evaluated all values between 0 and 1 with a step of 0.05. For rule 3 we evaluated all values between 0 and 200 with a step of 0.1 for values between 0 and 1 and a step of 1 for values between 1 and 200. Table 2 provides an overview of the performance of this measure when using a threshold of 0.7, which was, using a hill-climbing procedure, found to be the optimal threshold.

The rationale behind rule 1 is that tags are often used in both the singular and the plural form like ‘car’ and ‘cars’. The rule should ensure that these syntactic variations are marked correctly. The second rule aims at identifying syntactic variations that make use of non-alphanumerical characters as hyphenation, e.g., ‘high-speed’ and ‘highspeed’. In our experiments we found that the Levenshtein distance causes many errors when trying to identify syntactic variations in tags with numerical characters. To address this problem we have introduced rule 3. Numbers in tags often represent precise meanings like dates. We found

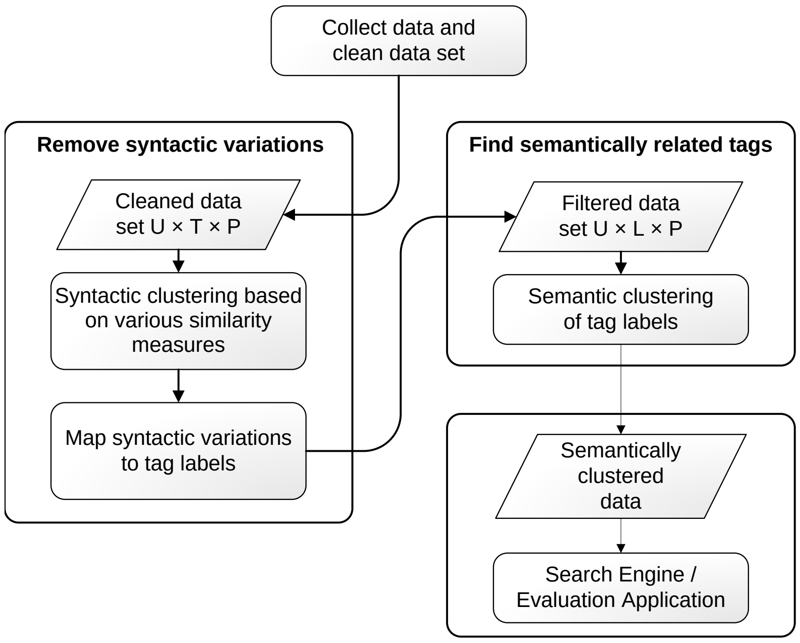
Collect data and clean data set
Remove syntactic variations
Cleaned data set U × T × P
Syntactic clustering based on various similarity measures
Map syntactic variations to tag labels
Find semantically related tags
Filtered data set U × L × P
Semantic clustering of tag labels
Semantically clustered data
Search Engine / Evaluation Application

Fig. 3. Overview of the framework

the Levenshtein distance produced a lot of errors with these type of tags because often there is a large difference in meaning between two tags while there is only a single number that is different. Consider the two tags '23062009' and '23012009' for example. The normalized Levenshtein similarity between these two strings is 0.875 which is relatively high while these strings do not represent syntactic variations. Finally, rule 4 applies the traditional Levenshtein cost to all other operations. The normalized variable cost Levenshtein similarity between tag i and j, denoted by $vclv_{ij}$, is defined in Equation 4.

$$vclv_{ij} = 1 - \frac{avclv_{ij}}{\max(length(t_i), length(t_j))} \tag{4}$$

3.2.3 Co-occurrence Data and the Cosine Similarity

To measure the semantic relatedness between tags and the syntactic similarity of short tags, we utilize the cosine similarity measure based on co-occurrence vectors of tags. This measure essentially describes the similarity of the context in which two tags appear. We define the context of a tag by measuring how often the tag is used together with other tags, called the tag co-occurrence. The cosine similarity between two tags measures to what extent the two tags have a similar context by comparing the co-occurrence vectors of these tags.

3.3 STCS Framework

In this section we discuss the two layers of the framework in detail. An overview of the framework is presented in Fig. 3.

3.3.1 Removing Syntactic Variations

For the detection of syntactic variations we have selected the Levenshtein similarity measure because related work shows it performs better in detecting syntactic variations than the Hamming distance measure [11]. Later research [12] suggests that a fuzzy based similarity measure that uses variable costs performs better in identifying syntactic variations than the Levenshtein distance. Therefore, inspired by this work, we use the normalized variable cost Levenshtein similarity ($vclv_{ij}$) that we have previously described in this paper. Because the Levenshtein distance does not perform well for short tags, we use two alternative measures that combine the cosine similarity, first, with the normalized Levenshtein similarity and, second, with the normalized variable cost Levenshtein similarity, respectively.

The measure combining the cosine similarity and the normalized Levenshtein similarity (CosLev_{ij}) is defined as follows:

$$\text{CosLev}_{ij} = z_{ij} \times lv_{ij} + (1 - z_{ij}) \times \cos(i,j) \tag{5}$$

where

$$z_{ij} = \frac{\max(\text{length}(t_i), \text{length}(t_j))}{\max_{t_k}(\text{length}(t_k))}, \text{ with } t_i, t_j, t_k \in T\ . \tag{6}$$

Using the normalized Levenshtein similarity for short tags may result in false positives, i.e., two tags being incorrectly identified as syntactic variations of each other, e.g., 'cat' and 'cut'. To address this problem the weight in the equation of the cosine similarity between two tags increases as the tags become shorter.

Initial experiments with the variable cost Levenshtein showed that a lower weight for the cosine similarity and a higher weight for the variable cost Levenshtein similarity improved the precision of the syntactic clustering. We therefore used a sample of 1100 tags with syntactic variations to determine a weight distribution that improves the recall and precision of the found syntactic variations. Based on these experiments we define the measure combining the cosine similarity with the variable cost Levenshtein similarity measure as follows:

$$\text{CosVarLev}_{ij} = (1 - vcz_{ij}) \times vclv_{ij} + vcz_{ij} \times \cos(i,j) \tag{7}$$

where

$$vcz_{ij} = \max(0, 0.3 - \frac{\max(\text{length}(t_i), \text{length}(t_j))}{\max_{t_k}(\text{length}(t_k))}) \tag{8}$$

with t_i, t_j, and $t_k \in T$.

For our data set, where the length of the longest tag in the data set is 32, this second combined measure only uses the cosine similarity when the length of the longest tag of the two tags is less than 10 characters. If the longest of the two strings contains 10 or more characters the measure fully relies on the normalized variable cost Levenshtein similarity and the cosine similarity is disregarded. The weight of the cosine similarity as related to the length of the longest of the two tags is depicted in Figure 4.

The framework uses an initial set of tag pairs with a normalized Levenshtein similarity or normalized variable cost Levenshtein similarity above a certain threshold α as input for the algorithm that removes syntactic variations. The α threshold represents the minimum similarity for which we consider two tags to possibly be syntactic variations of each other. This threshold is used to disregard tag combinations that are unlikely to be syntactic variations.

The initial list is used to create sets T and E as input to construct an undirected graph. The set T contains each unique tag from the input list. The set E is a set of weighted edges between the nodes in T, where the weight represents the similarity between corresponding tags as given by one of the two previously stated similarity measures.

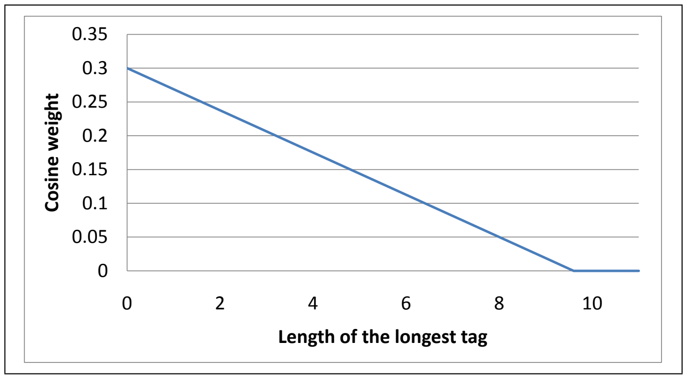

Fig. 4. The distribution of the cosine weight

To build the graph, all the elements from set T are first added as nodes to the graph. Next, all the edges from set E that have a weight above a certain threshold β are added to the graph, connecting the nodes from set T. A node is considered a syntactic variation of another node when there is an edge connecting the two nodes with a weight higher than β.

The syntactic clusters can then be obtained by retrieving the connected components in the graph as sets of vertices. A connected component is defined as a maximal subgraph in which all pairs of vertices in the subgraph are, directly or indirectly, reachable from one another. Each subgraph then contains all the nodes (tags) that are syntactic variations of each other and thus forms a cluster of syntactic variations. An example of some resulting subgraphs (clusters) is presented in Fig. 5 with each node containing a unique tag.

We utilize this data to create a new data set in which the tags from each cluster are aggregated and presented as a single tag, which we call the label for the cluster. The tag from a cluster that is used most frequently in the data set is selected as the label of that cluster.

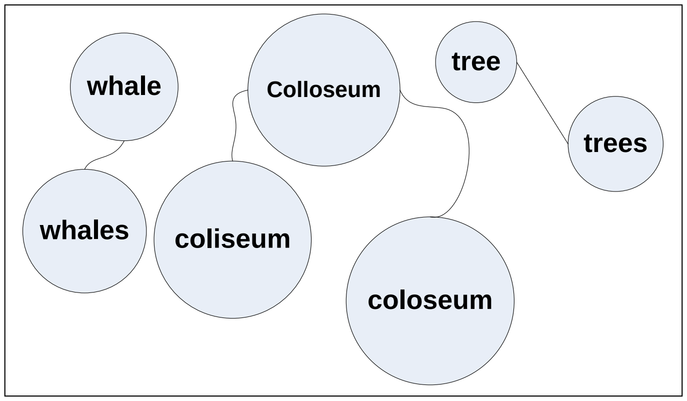

Fig. 5. An example of graph containing three subgraphs (clusters)

The resulting data set is used as input for the semantic clustering algorithm. An example of a syntactic cluster is shown in Table 6. The tag on the first row is the label for the cluster.

holland
hollande
hollandia
hollands
holanda
hollanda
holandia

Fig. 6. An example of syntactic variations

3.3.2 Clustering Semantically Related Tags

After the syntactic variations and misspellings have been removed from the dataset, the new dataset can be used to effectively construct semantic clusters. The Clustering by Committee [24] partitional clustering algorithm used by Specia and Motta [28] is employed for this purpose, both with and without some modifications. The algorithm was selected because it uses all tags instead of the cluster centroid as other partitional algorithms (e.g., k-means) do in order to calculate the similarity between two clusters. The algorithm also allows for a tag to appear in multiple clusters, representing thus its different meanings.

The algorithm first creates initial clusters where each tag is a separate cluster. Then all the tags for which the average cosine similarity with respect to all the tags in each cluster is above a certain threshold (χ) are added to the cluster. This often results in many identical or nearly identical clusters. Therefore, Specia and Motta utilize two smoothing heuristics that merge very similar and redundant clusters. For each combination of two clusters the algorithm checks if one cluster contains the other, i.e., if all the elements in the smaller cluster are also part of the larger cluster. If this is the case, the smaller cluster is removed. For each pair of clusters the algorithm also evaluates how similar the clusters are in order to merge clusters that are very similar. This is done by checking if the number of unique tags in the smaller cluster (with respect to the larger cluster), represents less than a fixed percentage of the number of tags in the smaller cluster. If this is the case the tags in the smaller cluster that do not occur in the larger cluster are added to the larger cluster and the smaller cluster is removed.

The problem with the second heuristic is that the percentage used for merging two similar clusters is fixed. Therefore, the minimum required number of different elements increases with the size of the smaller cluster. This is a problem because it makes it difficult to select a threshold for which the larger clusters do not merge too easily and the smaller clusters too difficultly. The minimum number of different elements for two clusters to prevent being merged, is given by the following function: $f(|c|) = \lfloor \epsilon \cdot |c| \rfloor$ where ϵ is the threshold and $|c|$ is the number of elements in the smaller cluster. So for $\epsilon = 0.40$ and $|c| = 30$, the minimum number of different elements is given by $f(30) = 12$. This means that cluster c will be merged into a larger cluster C, if $|D| \leq 12$ where $D = c - C$. Since $f(2) = 0$, a cluster with a size smaller than 3 is never merged.

To address the above described problem, we utilize two new heuristics, instead of the second heuristic. The first new heuristic considers the semantic relatedness between the large cluster and the cluster difference between the small and the large cluster. The second new heuristic considers the size of the difference between two clusters relative to the size of the smaller cluster in combination with a dynamic threshold.

The first new heuristic merges two clusters C and c where $|C| \geq |c|$ when the average cosine between all elements in cluster difference D and the elements in the large cluster is above a certain threshold δ. This average cosine is defined as:

$$\text{Average cosine} = \sum_{d \in D} \frac{Avg_d}{|D|} \tag{9}$$

where:

$$Avg_d = \sum_{x \in C} \frac{\cos(x, d)}{|C|} \tag{10}$$

The second new heuristic merges two clusters when the normalized difference between the clusters is smaller than the new dynamic threshold ϵ. The normalized difference η is defined as:

$$\eta = \frac{|D|}{|c|} \tag{11}$$

Threshold ϵ is defined as:

$$\epsilon = \frac{\phi}{\sqrt{|c|}} \tag{12}$$

thus:

$$f(|c|) = \lfloor \epsilon \cdot |c| \rfloor = \lfloor \phi \cdot \sqrt{|c|} \rfloor \tag{13}$$

The distribution of the maximum allowed difference for which two clusters are merged can be optimized by changing ϕ. If any of the three heuristics (the first old heuristic and the two new heuristics) are fulfilled we merge the considered clusters.

4 Framework Implementation

In this section we present our implementation of the framework described in the previous section. First we discuss the data collection and processing. Then, we describe the implementation of the two framework layers, i.e., removing syntactic variations and identifying semantic clusters. The implementation of the framework is done in Java in combination with a MySQL database. For data collection and processing we used PHP scripts.

4.1 Data Processing

In this section we discuss the collection of our data set. We also explain which data processing steps we have performed in order to obtain the data set that we used for the evaluation.

4.1.1 Data Collection

We gathered data from the popular picture sharing website Flickr.com for our experiments. For the original data set we collected all the (public) pictures uploaded to Flickr in 2009, together with their associated tags and users. To speed up the data collection we distributed the process over four machines. Our initial data set contained 38,788,518 pictures, 1,017,168 tags, and 196,344 users. To make the data set suitable for our experiments we performed a data cleaning operation.

4.1.2 Data Set Cleaning

After gathering the data we used a few procedures to clean it. Flickr has very few restrictions for users with regards to how they tag their pictures, which is a source of noise in our data set. We applied the following cleaning steps in the order in which they are listed:

(1) *Remove tags with a tag length larger than 32 characters.*
The data set contained tags consisting of multiple tags or words without spaces that formed complete sentences. These were removed as we solely want individual tags (words) in our data set.

(2) *Remove tags containing unrecognizable signs.*
Non-Latin characters (like Arabic or Cyrillic signs), signs which are not part of the Latin alphabet or of the numeric signs, are removed because it is impossible for us to evaluate the resulting clusters.

(3) *For each user, remove images with identical tag strings.*
One of the problems we encountered in our data set was the presence of hundreds of pictures uploaded by the same user with identical tags. These were, for example, sets of holiday pictures annotated with identical tags, often unrelated to the picture. Someone would for example tag hundred pictures of a trip to France with 'eiffeltower' and 'beach' simply because some of the pictures contained the Eiffel Tower and others pictured a beach. To prevent these sets from influencing the co-occurrence measure, and in the end our clustering results, we only kept one arbitrary picture of each of these sets and removed the others.

(4) *Remove tags which occur in less than 133 different pictures.*
We used the statistic $Q_1 - 1.5 \times IQR$ to identify outliers and found that tags which were used less than 133 times should be considered as outliers. We do not remove frequently occurring tags, because these offer valuable information.

After the initial cleaning the data set contained 147,064,188 associations, 31,951,884 co-occurrence pairs, and 97,569 tags.

4.1.3 Calculating Cosine Similarity

In order to calculate the cosine similarity between two tags, the co-occurrence vector for each tag is required. We obtain these vectors by constructing a matrix, with both a row and a column for each tag, in which the cells contain the co-occurrence for that particular combination of tags. We used the matrix data structure from the Colt library [2] to store the co-occurrence matrix because it only stores non-zero values in memory, resulting in a small memory footprint. Even though Colt is designed for large matrices, it was unable to handle the large size of our data sets. We addressed this problem by implementing our own high performance matrix library on top of Colt, which utilizes a Colt vector to store each column of the matrix. This means that in our implementation a matrix with n columns utilizes n Colt vectors, each representing a column. We used this matrix to calculate the cosine similarity for each unique combination of two tags in our data set.

Using the previous matrix representation we encountered two problems. First the matrix was very large in size and the machines we had available, which contained up to 6 GB

RAM, did not have enough memory available to store the matrix. The second problem we encountered was that the number of calculations we needed to perform was very large and it would take a single machine infeasibly long to complete them. We tried to address the first problem by storing the part of the matrix the algorithm was currently operating on in memory and the rest on the hard drive, loading the required parts from the hard drive just-in-time for the needed computations. This made each calculation a lot slower making our second problem larger, because the total required computation time increased.

Both previously described problems were addressed by utilizing the cloud computing service Amazon EC2 [3]. We implemented the algorithms in a distributed architecture and executed them in parallel on multiple Amazon EC2 high memory instances each having 17.1 GB RAM and thus being capable to load the entire matrix in memory. Using Amazon's auction system we placed bids on the overcapacity in their system. Based on our bids, other bids, and available oversupply, the number of instances running in parallel was updated every 30 minutes automatically. The total number of instances running in parallel fluctuated trough time between 3 and 52 instances. Due to the use of the α threshold, for the syntactic clustering we performed 5,829,400 cosine similarity computations using 16 computing hours. The computations were finished by the Amazon EC2 system in 2.5 hours.

In order to reduce the time required for the cosine calculations for the semantic clustering process, we only considered the 10,000 most popular tags for the semantic clustering. However, we made sure that the co-occurrence information of these tags compromises all the information provided by the associations made in 2009 that contain one of these 10,000 tags (approximately 72% of the total amount of associations). For the semantic clustering we performed 150,000,000 cosine similarity computations in 33 hours using 192 computing hours. When Amazon EC2 launched a new instance our software was automatically started. The total of 150,000,000 cosines is needed because there are three syntactic clustering methods used in the semantic clustering (normalized Levenshtein, normalized Levenshtein + cosine, and variable cost Levenshtein + cosine) and each syntactic clustering method requires 50,000,000 cosines to be computed ($3 \times 50{,}000{,}000 = 150{,}000{,}000$). The reason why each syntactic clustering method that relies on the cosine similarity requires the 50,000,000 cosines to be computed is that it is the number of cosine combinations in the upper (or lower) triangular matrix of tag combinations, i.e., $(10{,}000 \times 10{,}000 - 10{,}000)/2 = 50{,}000{,}000$.

Each Amazon EC2 instance loaded the full matrix from the disk in RAM and connected to a central job server which coordinated each instance to perform a distinct portion of the calculations. When an instance finished its computations, it sent the results back to the job server which updated the central MySQL database with the results and sent a new job to the instance. An overview of this process can be seen in Fig. 7.

The disadvantage of using the overcapacity in the Amazon EC2 system is that the instances can be terminated at any moment. Therefore, there was a risk that the job server would receive corrupted results from an instance which was terminated while sending the results of its computations back to the job server. To address this issue we had a validation program running on the job server which validated the incoming results by verifying if each result contained two tag id's and a cosine similarity and by checking if the number of returned results matched the number of calculations in the job the instance was working on. If there was a mismatch the results were disregarded and the job was rescheduled to be calculated by

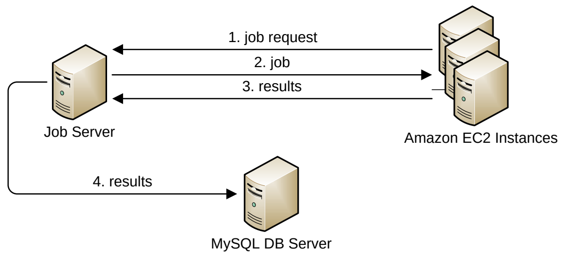

Fig. 7. Overview of the distributed calculations process

a different instance.

4.1.4 Calculating (Variable Cost) Levenshtein Distance and the Combined Measure

We used the SimMetrics [4] library to perform the Levenshtein Distance measure calculations because it provided an efficient and easy to use implementation of this measure. We then modified the SimMetrics library to support the variable cost Levenshtein Distance measure. We did not encounter any performance problems and applied the calculation to every cell of our co-occurrence matrix storing the resulting values in a MySQL database. Finally we used a Java program which operated on the database with the precomputed cosine similarity measure and Levenshtein measure to compute the combined measure.

4.2 Removing Syntactic Variations and Semantic clustering

We used the Java Universal Network/Graph Framework (JUNG) [5] graph library to store the graph used by the algorithm that performed the syntactic clustering. We chose this library because it provides an efficient and easy to use method to retrieve the sets of connected components, in our case clusters, from a graph. The graph was populated by loading the nodes and edges from the MySQL database in which we also stored the resulting syntactic clusters. We used a separate Java program operating on this database to aggregate the co-occurrence information of the tags identified as syntactic variations. The semantic clustering algorithms are also implemented in Java.

5 Results and Discussion

In this section we present the experimental results obtained using the framework implementation. In the next sections, we discuss the evaluation of each of the two framework layers, i.e., removing syntactic variations and identifying semantic clusters.

5.1 Removing Syntactic Variations

We chose $\alpha = 0.5$ as a threshold for the normalized (variable cost) Levenshtein similarity to identify potential syntactic variations. We obtained this value using a sample of 100 tag pairs that were known to be syntactic variations. We found that the normalized (variable cost) Levenshtein similarity between these tag pairs was never smaller than 0.5. The goal of this threshold was to reduce the number of potential syntactic variations for which the calculation of the cosine similarity was required. A value of 0.5 effectively reduced this number without possibly losing syntactic variations.

For the identification of syntactic variations we evaluated the following similarity measures:

- normalized Levenshtein similarity;
- normalized variable cost Levenshtein similarity;
- measure combining the normalized Levenshtein similarity with the cosine similarity, called CosLev;
- measure combining the normalized variable cost Levenshtein similarity with the cosine similarity, called CosVarLev.

For each of these measures the β thresholds that we have used can be found in Table 8. We have chosen these values because they resulted in the best precision on random samples of 100 clusters. For this threshold we tried all values between 0 and 1 with a step of 0.1 and all values between 0.85 and 0.95 (where the best results were found) with a step of 0.025. For the evaluation of each value we drew a random sample of 100 clusters (our training set).

Similarity measure	β
normalized Levenshtein	0.7
variable cost Levenshtein	0.875
CosLev	0.7
CosVarLev	0.85

Fig. 8. Similarity measures and the β threshold

We evaluated the performance of the different measures by drawing a different random sample of 100 clusters (our test set) for each measure and evaluating these manually. The evaluation of each cluster in the random samples was done using majority voting with a group of 3 users. Each person in the group chose the number of correct tags in a cluster and majority voting is then used to determine the final number of correct tags that is used in the precision and purity calculations. The results of this evaluation can be found in Table 9, which shows the average precision and the purity measure for the clusters obtained using each of the four similarity measures.

By performing a one-tailed unpaired unpooled two sample t-test with a significance level of 0.05, we can conclude that the measure combining the normalized Levenshtein similarity with the cosine similarity (CosLev) performs significantly better than the normalized Levenshtein similarity alone in terms of precision per cluster. The associated p-value for this t-test is 0.0001. In the same way, we can conclude that the measure combining the normalized variable cost Levenshtein similarity with the cosine similarity (CosVarLev) performs significantly better than the normalized variable cost Levenshtein similarity alone. The associated p-value is 0.03. With a p-value of 0.02, we can also conclude that the CosVarLev measure performs significantly better than the CosLev measure. Because the combined measures resulted in the best performance, we used these measures to identify the syntactic variations before moving on to the semantic clustering layer. Using the CosLev similarity measure resulted in the identification of 9,373 syntactic variations, while the CosVarLev measure identified 13,750 syntactic variations.

Some examples of resulting clusters with syntactic variations are presented in Table 10. The tags on the first row are the labels for the clusters. For the cluster 'kitten' we can see that it contains the plural form ('kittens'). Also, we observe that the cluster 'polizei' contains syntactic variations that are translations in various languages, e.g., 'politie' and 'polis'.

Similarity measure	**Average precision**	**Purity**
normalized Levenshtein	0.70	0.67
variable cost Levenshtein	0.89	0.84
CosLev	0.89	0.88
CosVarLev	0.94	0.92

Fig. 9. Similarity measures performance

kitten	**piccadilly**	**polizei**
kittens	picadillycircus	politi
kitty	picadilly	polizia
kittie	piccadillycircus	politie
kitties		policia
kitteh		polis
		poliisi
		policja

Fig. 10. Examples of clusters with syntactic variations

5.2 Finding Semantically Related tags

For the evaluation of the semantic clustering process, we have experimented with different combinations of similarity measures used for the syntactic clustering process and clustering methods used for the semantic clustering process. An overview of these combinations can be found in Table 11.

Method #	**Similarity used for syntactic**	**Algorithm used for semantic**
0	normalized Levenshtein	Specia and Motta w/ original heuristics
1	CosLev	Specia and Motta w/ original heuristics
2	CosLev	Specia and Motta w/ new heuristics
3	CosVarLev	Specia and Motta w/ new heuristics

Fig. 11. Clustering methods

All semantic clustering algorithms use the cosine similarity based on co-occurrence vectors as a similarity measure. For the different semantic clustering methods we choose the thresholds as displayed in Table 12. We have selected these thresholds because they resulted in the highest precision on random samples of 50 clusters (our training set). For each threshold we drew a separate random sample for all values between 0.1 and 0.9 with a step of 0.1. For the evaluation of the clusters in these random samples we again used majority voting with a group of 3 users. For the chosen thresholds we used majority voting and a different random sample of 100 clusters (our test set) for each method to calculate the average precision and purity, which can be found in Table 13.

Method #	χ	δ	ϕ	ϵ
0	0.8	–	–	0.2
1	0.8	–	–	0.3
2	0.8	0.7	0.9	–
3	0.8	0.7	0.7	–

Fig. 12. Thresholds used

Method #	**Average precision**	**Purity**
0	0.75	0.73
1	0.80	0.78
2	0.86	0.89
3	0.93	0.92

Fig. 13. Evaluation results

Again we performed a one tailed unpaired unpooled two sample t-test with a significance level of 0.05. From this, we can conclude that the methods using the new heuristics (methods 2 and 3) perform significantly better than the original Specia and Motta method (method 0) in terms of precision per cluster. The p-values for these tests are 0.0007 and 0.00007, respectively. With a p-value of 0.04 we can also conclude that, when using the CosLev similarity for the syntactic clustering, and using the new heuristics for the semantic clustering algorithm (method 2) significantly improves the precision per cluster when compared to using the original heuristics for the semantic clustering and CosLev for syntactic clustering (method 1). Last, we can conclude, based on a p-value of 0.01, that using the CosVarLev similarity for the syntactic clustering phase (method 3) significantly improves the precision per cluster of the semantic clustering algorithm when compared to using the CosLev similarity for the syntactic clustering phase (method 2).

5.3 Picture Search Engine

To evaluate if the proposed clustering methods improve the search and exploration of tag spaces, we have built a picture search engine. The picture search engine allows users to search our entire data set using one of the previously defined clustering methods. Users can enter a query consisting of one or multiple tags separated by spaces. After submitting a query, the results are presented to the user. Any additional information provided by the clustering algorithms is presented in the left column on the results page. An image of the results page for the query ‘surf’ is shown in Figure 14. In this image we can see on the left side the identified clusters for the query ‘surf’. We now first explain how the search engine works and then we present the results of the evaluation using the search engine.

5.3.1 Implementation

The picture search engine processes user queries by querying our entire dataset of 38,788,518 images using any of the 97,568 tags from our filtered data set. The user can choose from the different clustering methods available which are “Original”, “Original + CosLev”, “Modified + CosLev” and “Modified + CosVarLev”. “Original” is the original method as proposed by Specia and Motta, which uses the normalized Levenshtein similarity for the syntactic

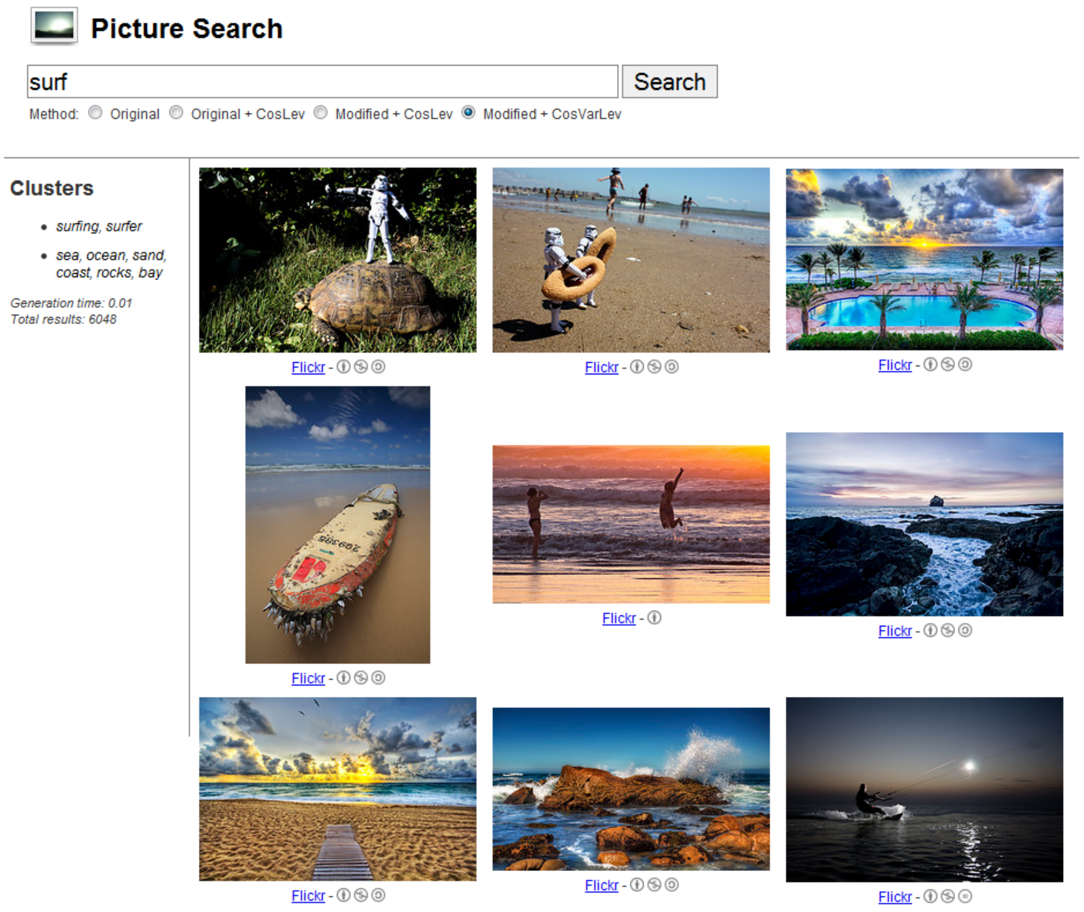

Fig. 14. The results page for the query ‘surf’

clustering. “Original + CosLev” is the original semantic clustering algorithm as proposed by Specia and Motta enhanced with the CosLev combined measure for syntactic clustering. The “Modified + CosLev” option utilizes the modified version of the Specia and Motta algorithm and the CosLev measure for the syntactic clustering. Finally the “Modified + CosVarLev” option uses the modified Specia and Motta algorithm and the new combined measure which is based on the variable cost Levenshtein and cosine similarity to perform syntactic clustering.

Retrieving and Sorting the Results The search engine uses a relatively straightforward approach to retrieve and sort the results of a query. To retrieve the results of a query the system checks which images have been tagged with the keywords in the query. These pictures are sorted based on the cosine similarity between the query tags and the image tags. For equal cosine values, these pictures are then sorted based on the number of views each picture received on Flickr.com in the period before we retrieved our dataset from Flickr.

Syntactic Variations When a users enters a tag that we have identified as a syntactic variation, the system automatically replaces this tag internally with its label. This means that when a users searches for 'holanda', the system will also include pictures tagged with 'holland', 'hollande', 'hollandia', 'hollands', 'hollanda' and 'holandia' in the result. This improves the search results because it allows the system to correct for syntactic variations and possibly show more relevant results, i.e., it improves recall.

Semantic Clusters When the system detects that a particular tag is part of multiple clusters, these clusters are presented to the user in addition to the results of the query. The user can select a specific cluster which is utilized by the system to reorder the results of the query showing more relevant results, based on the selected cluster, first. When a user selects a specific cluster the search engine sorts the resulting images descending based on the cosine similarity between the tags assigned to each image and the tags in the chosen cluster. This feature allows the user to obtain more relevant results when searching for a picture using a tag that has multiple meanings. We define the cosine similarity between an image y and a cluster x as follows:

$$\cos(x, y) = \frac{x \cdot y}{\|x\|\|y\|} \tag{14}$$

where x is the tag vector of the image, which contains a binary value for each tag which is 0 if the tag has not been associated with the image and 1 if the tag has been associated with the image, and y is the tag vector of the cluster which contains a binary value for each tag which is 0 if the tag is not part of the cluster and 1 if the tag is part of the cluster. When a query contains multiple tags that each are present in separate clusters, the system enables the user to select one of the available clusters for each tag. To sort the results of this query the cosine similarity between each image and each individual selected cluster is calculated and the images are sorted descending based on the average cosine similarity between the images and the clusters.

Related Tags When a tag occurs in a single semantic cluster we utilize this cluster information to provide the user with a list of related tags. The user can select one of the related tags to explore related pictures. When a related tag is selected the system shows the results for this related tag. If the original query consists of multiple tags, a query tag can be replaced by its related tag. This feature is useful for example when the results of a query do not provide the user with any results. The list of related tags can then assist the user in finding results for similar queries. Such a feature can also be useful when there are too many results. By selecting a related tag, the user has to browse through a smaller selection of pictures.

5.3.2 Evaluation

The evaluation of the search engine is done using a random sample of 50 tags. Again majority voting with a group of 3 users is used. For the evaluation of the syntactic variations detection we use these 50 tags as queries and determine the precision for the first 20 results (P@20). We also compute the mean average precision (MAP) using samples of the first 20 results for each query. We measure recall by counting the number of correctly retrieved pictures among the first 20 results. Better identification of syntactic variations should lead to a higher average P@20 and higher mean average precision, because the search algorithm can include more correct syntactic variations of the entered query in the search. The results of this evaluation can be seen in Table 15.

Similarity measure	**Average P@20**	**MAP**	**Average # images correctly retrieved**
normalized Levenshtein	0.84	0.87	16.74
CosLev	0.86	0.88	17.1
CosVarLev	0.92	0.93	18.28

Fig. 15. Syntactic variation detection

By performing a one tailed paired t-test with a significance level of 0.05 we can conclude that the CosVarLev similarity measure significantly improves the precision P@20 when compared with the normalized Levenshtein similarity. The associated p-value is 0.01. These results partly confirm the syntactic clustering evaluation conclusions from Section 5.1. However, from this small sample we cannot conclude the CosVarlev similarity significantly improves the P@20 when compared with the CosLev similarity or that the CosLev similarity significantly improves the P@20 when compared with the normalized Levenshtein similarity.

To evaluate the effect of the different semantic clustering methods mentioned in Section 5.2 on the searching and retrieving of pictures we use a set of 16 ambiguous tags as queries. We evaluate the performance of the techniques by comparing the P@20 and MAP obtained using the different methods. For this purpose we allow users to select clusters to better specify their queries and evaluate the results obtained based on the users final selections. The results of this evaluation can be found in Table 16.

Semantic clustering method #	**Average P@20**	**MAP**	**Average # images correctly retrieved**
0	0.36	0.41	7.25
1	0.42	0.43	8.31
2	0.53	0.57	10.63
3	0.64	0.65	12.81

Fig. 16. Semantic clustering search evaluation

Because of the non-normality of the data and the small sample size, we cannot use a t-test. Therefore, we now use the non-parametric Wilcoxon signed rank test for two paired samples. Using this test with a significance value of 0.05 we can conclude that method 3, in terms of P@20, performs significantly better than method 0, method 1, and method 2, with p-values of 0.002, 0.003, and 0.005, respectively. Thus, the new heuristics in combination with the CosVarLev measure leads to better results than using the original heuristics and the normalized Levenshtein or CosLev measure.

6 Conclusion

The main objective of this paper is to propose a method to perform syntactic and semantic tag clustering. For the syntactic clustering we compare a measure that combines the normalized Levenshtein similarity and the cosine similarity with a measure that combines the normalized variable cost Levenshtein similarity with the cosine similarity. Our conclusion is that the measure using the variable cost Levenshtein similarity and the cosine similarity performs significantly better with regard to precision than the other measure. Using this measure the clustering method as proposed in the framework was also able to effectively find more syntactic variations from the data set.

For the semantic clustering we evaluated the state-of-the-art algorithm proposed by Specia and Motta [28] both with and without two new heuristics for merging two similar clusters. The first new heuristic considers the semantic relatedness between the large cluster and the cluster difference between the small and the large clusters. The second new heuristic considers the size of the difference with respect to the small cluster in combination with a dynamic threshold. We also investigate the impact of the different syntactic and semantic clustering methods. Our conclusion is that the method using the new heuristics and the normalized variable cost Levenshtein similarity combined with the cosine similarity as a similarity measure for the syntactic clustering process, significantly outperforms the other methods in terms of precision. We have also shown that our results are valid on a significantly larger dataset than was used in similar works (38,738,518 resources).

In the future we would like to experiment with the use of the Wikipedia [6] as a tool to help identify syntactic variations of tags and to detect different meanings of a tag. Wikipedia might prove useful because, like tags, the content on Wikipedia is generated by users. This means that Wikipedia is more likely to (quickly) include new words, new senses of words or syntactic variations of words than other providers of lexical information, like WordNet [22].

We would also like to investigate how we can further improve the concept of the variable cost Levenshtein measure. For this purpose we plan to use a machine learning algorithm to develop a variable cost Levenshtein distance measure. We want to leverage the machine learning algorithm to estimate for each operation in the Levenshtein algorithm the probability that this operation constitutes a syntactic variation. As algorithm variables we would like to consider: the characters involved in the operation, the position of the characters in the string, the adjacent characters, the length of the strings, and the type of operation, i.e., insert, substitute, or delete. Additionally, we would also like to experiment with other clustering methods based on the minimum description length principle [30].

References

1. Flickr Online Photo Sharing Service: `http://www.flickr.com`.
2. Colt Libraries for High Performance Scientific and Technical Computing in Java: `http://acs.lbl.gov/~hoschek/colt/`.
3. Amazon Elastic Compute Cloud (Amazon EC2): `http://aws.amazon.com/ec2`.
4. SimMetrics Java Library: `http://www.dcs.shef.ac.uk/~sam/simmetrics.html`.
5. Java Universal Network Graph (JUNG) Framework: `http://jung.sourceforge.net`.
6. Wikipedia Online Encyclopedia: `http://en.wikipedia.org`.
7. G. Begelman, P. Keller, and F. Smadja. Automated tag clustering: Improving search and exploration in the tag space. In *Collaborative Web Tagging Workshop (WWW 2006)*, pages 22–26,

2006.
8. C. Cattuto, D. Benz, A. Hotho, and G. Stumme. Semantic grounding of tag relatedness in social bookmarking systems. In *7th International Semantic Web Conference (ISWC 2008)*, pages 615–631. Springer, 2008.
9. A. Dattolo, F. Tomasi, and F. Vitali. Towards disambiguating social tagging systems. In San Murugesan, editor, *Handbook of Research on Web 2.0, 3.0 and X.0: Technologies, Business, and Social Applications*, chapter 20, pages 349–369. IGI Global, 2010.
10. F. Echarte, J.J. Astrain, A. Crdoba, and J. Villadangos. Ontology of folksonomy: A new modeling method. In *Semantic Authoring, Annotation and Knowledge Markup Workshop (SAAKM 2007)*, pages 28–31. CEUR-WS, 2007.
11. F. Echarte, J.J. Astrain, A. Crdoba, and J. Villadangos. Pattern matching techniques to identify syntactic variations of tags in folksonomies. In *1st World Summit on The Knowledge Society (WSKS 2008)*, pages 557–564. Springer, 2008.
12. F. Echarte, J.J. Astrain, A. Crdoba, and J. Villadangos. Improving folksonomies quality by syntactic tag variations grouping. In *2009 ACM Symposium on Applied Computing (SAC 2009)*, pages 1226–1230. ACM, 2009.
13. S. Golder and B. A. Huberman. Usage patterns of collaborative tagging systems. *Journal of Information Science*, 32(2):198–208, 2006.
14. R.W. Hamming. Error detecting and error correcting codes. *Bell System Technical Journal*, 26(2):147–160, 1950.
15. J.J. Jiang and D.W. Conrath. Semantic similarity based on corpus statistics and lexical taxonomy. In *International Conference on Research in Computational Linguistics (ROCLING X)*, pages 19–33, 1997.
16. B. Larsen and C. Aone. Fast and effective text mining using linear-time document clustering. In *Fifth ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD 1999)*, pages 16–22. ACM, 1999.
17. V.I. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Physics Doklady*, 10(8):707–710, 1966.
18. C.D. Manning, P. Raghavan, and H. Schtze. *Introduction to Information Retrieval*. Cambridge University Press, 2008.
19. B. Markines, C. Cattuto, F. Menczer, D. Benz, A. Hotho, and G. Stumme. Evaluating similarity measures for emergent semantics of social tagging. In *18th World Wide Web Conference (WWW 2009)*, pages 641–650. ACM, 2009.
20. A. Mathes. Folksonomies - cooperative classification and communication through shared metadata, 2004. Computer Mediated Communication, LIS590CMC (Doctoral Seminar), Graduate School of Library and Information Science, University of Illinois Urbana-Champaign `http://www.adammathes.com/academic/computer-mediated-communication/folksonomies.html`.
21. D.R. Millen and J. Feinberg. Using social tagging to improve social navigation. In *Workshop on the Social Navigation and Community-based Adaptation Technologies (SNC-BAT 2006) at AH 2006*, pages 532–541, 2006.
22. G.A. Miller, R. Beckwith, C. Fellbaum, D. Gross, and K. Miller. WordNet: An on-line lexical database. *International Journal of Lexicography*, 3:235–244, 1990.
23. M.E.J. Newman and M. Girvan. Finding and evaluating community structure in networks. *Physical Review E*, 69(2):026113, 2004.
24. P. Pantel. *Clustering by Committee*. PhD thesis, University of Alberta, 2003. `http://www.patrickpantel.com/cgi-bin/web/tools/getfile.pl?type=paper&id=2003/cbc.pdf`.
25. J. Radelaar, A.J. Boor, D. Vandic, J.W. van Dam, F. Hogenboom, and F. Frasincar. Improving the exploration of tag spaces using automated tag clustering. In *Eleventh International Conference on Web Engineering (ICWE 2011)*, volume 6757 of *Lecture Notes in Computer Science*, pages 274–288. Springer, 2011.
26. M. Sanderso. and B. Croft. Deriving concept hierarchies from text. In *22nd ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR 1999)*, pages 206–213. ACM,

1999.

27. P. Schmitz. Inducing ontology from flickr tags. In *Collaborative Web Tagging Workshop (WWW 2006)*, pages 206–209, 2006.
28. L. Specia and E. Motta. Integrating folksonomies with the semantic web. In *4th European Semantic Web Conference (ESWC 2007)*, pages 503–517. Springer, 2007.
29. J.W. van Dam, D. Vandic, F. Hogenboom, and F. Frasincar. Searching and browsing tag spaces using the semantic tag clustering search framework. In *4th International Conference on Semantic Computing (ICSC 2010)*, pages 436–439. IEEE, 2010.
30. M. van Leeuwen, F. Bonchi, B. Sigurbjrnsson, and A. Siebes. Compressing tags to find interesting media groups. In *18th ACM Conference on Information and Knowledge Management (CIKM 2009)*, pages 1147–1156. ACM, 2009.
31. D. Vandic, F. Frasincar, and F. Hogenboom. Scaling pair-wise similarity-based algorithms in tagging spaces. In *12th International Conference on Web Engineering (ICWE 2012)*, volume 7387 of *Lecture Notes in Computer Science*, pages 46–60. Springer, 2012.
32. D. Vandic, J.W. van Dam, and F. Frasincar. A Semantic-Based Approach for Searching and Browsing Tag Spaces. *Decision Support Systems*, 54(1):644–654, 2012.
33. D. Vandic, J.W. van Dam, F. Hogenboom, and F. Frasincar. A semantic clustering-based approach for searching and browsing tag spaces. In *26th Symposium on Applied Computing (SAC 2011)*, pages 1693–1699. ACM, 2011.
34. R. Vermaas, D. Vandic, and F. Frasincar. Incremental cosine computations for search and exploration of tag spaces. In *22nd Database and Expert Systems Applications (DEXA 2012)*, volume 7447 of *Lecture Notes in Computer Science*, pages 156–167. Springer, 2012.
35. C.A. Yeung, N. Gibbins, and N. Shadbolt. Contextualising tags in collaborative tagging systems. In *20th ACM Conference on Hypertext and Hypermedia (HT 2009)*, pages 251–260. ACM, 2009.