Economics & Informatics

Erasmus School of Economics

Erasmus University Rotterdam

Bachelor Thesis

A Semantic-Based Approach for Searching and Browsing Tag Spaces

Authors:

Jan-Willem van Dam
305069

Damir Vandic
305415

Supervisor:

Dr. Flavius Frasincar

Co-supervisor:

Frederik Hogenboom MSc.

14th of July 2009

Abstract

In this thesis we propose the Semantic Tag Clustering Search framework (STCS). This framework consists of three parts. The first part deals with syntactic variations by clustering tags that are syntactic variations of each other and assigning a label to them. The second part of the framework addresses the problem of recognizing homonyms and identifying semantically related tags. The last, and final part of the STCS framework utilizes the clusters obtained from the first two parts to improve search and exploration in tag spaces. For removing syntactic variations, we use the normalized Levenshtein distance, and the cosine similarity measure based on tag co-occurrences. For creating semantic clusters, we employ two non-hierarchical, and two hierarchical clustering techniques. To evaluate the value of the semantic clusters, we develop an Web Application called XploreFlickr.com.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

There are a lot of Web services where users can employ tags to label content on the Web. Flickr [1] and Delicious [2] (also known as del.icio.us) are two well-known applications which make use of tags. In this thesis we focus on the Flickr service. Users that are registered on the Flickr Web site can upload photographs and assign tags to them. As with most tagging systems the user has no restrictions on the tags that can be used, the user can use any tag he or she wants.

Though tags are a flexible way of categorizing data, there are some limitations when considering search and exploration in tagging systems. Because the users have a lot of freedom in tagging resources, they can, for example, make typographical errors or use syntactic variations. This results in having different tags having the same meaning. An example of a typographical mistake would be to use 'selfportait' instead of 'self portrait'. Figure 1.1 and Figure 1.2 show what the consequences of these mistakes are when an user searches for these tags. Searching for 'self portrait' gives us $1,265,127$ more results than searching for 'selfportait'. Examples of a syntactic variations would be among other things plurals and singulars, like 'self portraits' and 'self portrait'. These typographical errors and syntactic variations of tags are important aspects to consider when designing a search engine. Google, for example, has auto syntactic variation detection in their search service. Often you get *'Did you mean...'* from Google where the engine tries to suggest the correct search term.

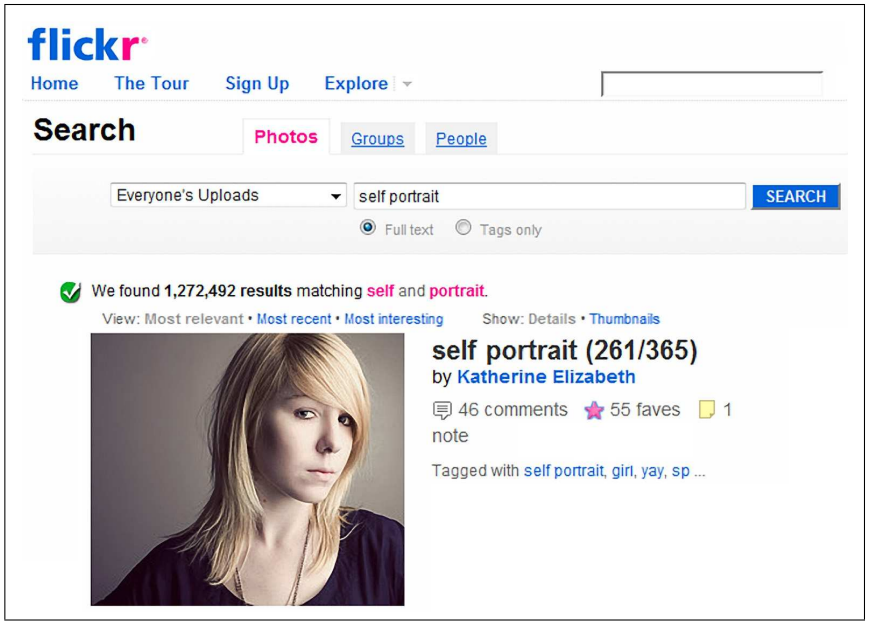


Figure 1.1: Searching for 'self portrait' gives $1,272,492$ results

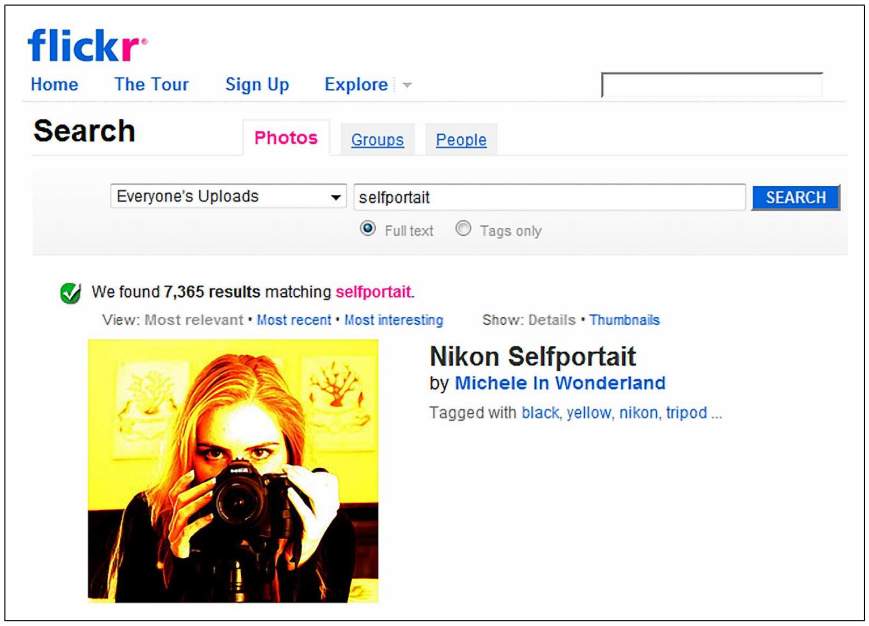


Figure 1.2: Searching for 'selfportait' gives $7,365$ results

Furthermore, people can use synonyms for the same concepts. These synonyms result in different results when people search, explore, or retrieve information from a tagging system like Flickr. For instance, somebody tags a picture

with ‘city’ where somebody else would have used ‘town’.

People also describe pictures differently. For a picture which shows the interior of a house, most people would use the tag ‘interior’, where others use a tag like ‘inside’ or ‘furniture’. These tags are semantically related. When people search for ‘furniture’ they are probably also interested in pictures tagged with ‘interior’. Another example of semantically related tags would be ‘Web 2.0’, ‘Ajax’, and ‘XML’.

Additionally, homonyms can occur. For instance, the word ‘apple’ is a homonym. When an user searches for ‘apple’, the search engine returns pictures related with both the brand ‘Apple’ as well as an apple and maybe other fruit. The search engine cannot distinguish between the multiple meanings the word ‘apple’ can have. Figure 1.3 illustrates this issue. Another example of a homonym is ‘rock’. Somebody can mean ‘rock’ as in ‘rock and roll’ or ‘rock’ as a stone.

When searching and exploring a tag space, the previously described symptoms (typographical mistakes, syntactic variations, synonyms, homonyms, and related tags) are a problem. The currently used search engines for tagging systems do not deal with these symptoms. In general, there are no structures, hierarchies, classifications, or clusters available in most tagging systems. A reason might be the enormous amount of data, if you do not structure incrementally. Better search engines for such tagging systems could be valuable for a lot of people, organizations, and companies. For example, marketing companies often need pictures in their daily activities and these companies would certainly benefit from more structured tagging systems. In this thesis we want to improve searching and exploring tag spaces by coping with syntactic variations, typographical mistakes, synonyms, homonyms, and related tags.

1.1 Problems and Goals

Our research goal is to gain insight into the possibilities of improving search and exploration in tag spaces, especially for marketing companies. Related work reveals that clustering techniques are appropriate for improving search and exploration in tag spaces. Thus, our main research question is

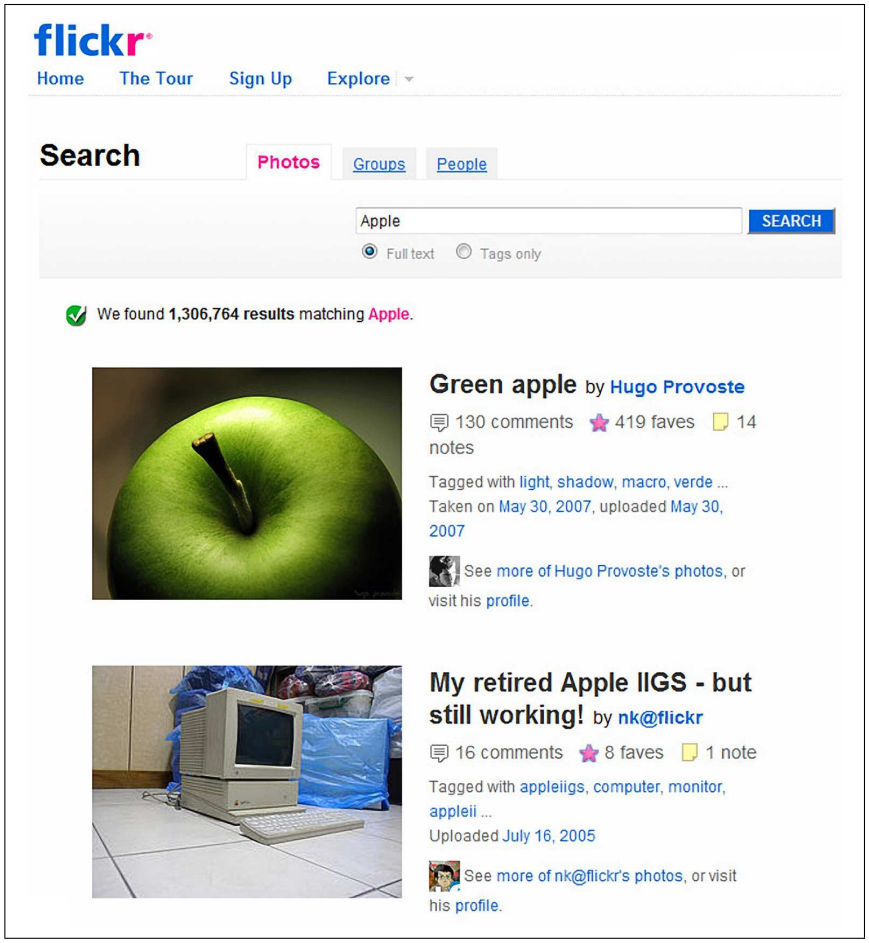


Figure 1.3: An example of a homonymous word

'How can one utilize clustering techniques to improve exploring and searching tag spaces?'

Related to this main research question the sub questions are defined as follows:

1. How can one deal with syntactic variations and typographical mistakes?

2. How can one deal with semantic symptoms (homonyms, synonyms, and related tags) in tagging systems ?

3. What kind of clustering techniques are best to be used for improving search and exploration in tagging systems?

1.2 Methodology

To address the research goal and problems we define an objective for every sub question as presented in Section 1.1:

1. An implementation for dealing with syntactic variations and typographical mistakes by using clustering techniques;
2. An implementation for dealing with semantic symptoms of tagging systems by using several clustering techniques;
3. Implementations for searching in tag spaces by using several clustering techniques for syntactic and semantic clustering.

To reach these objectives we design and implement an appropriate framework. The framework is called the Semantic Tag Clustering Search (STCS) framework. The framework consists of three parts, namely: a part where syntactic variations are identified, a part where semantic clusters are derived, and a part where one can search in tag spaces by using search methods utilizing these clusters.

In this thesis we consider two types of clustering methods, namely the non-hierarchical clusters and the hierarchical clusters. The non-hierarchical clusters contain related tags, but the there is no hierarchy in these type of clusters. On the contrary, hierarchical clusters do have these hierarchies, either for tags or clusters of tags.

For the non-hierarchical clusters we consider the method proposed by [3]. The reason why we have chosen to implement the method proposed in [3], is that their clustering algorithm allows tags to appear in multiple clusters. In this way homonyms could be easily detected and put in different contexts.

The method proposed by the authors of [4] are implemented for the hierarchical clusters. We have chosen for [4], because his method is proven better than that of [5] and [6]. An alternative could have been the method in [7]. This method in [7] is meant for *collaborative* tagging systems, like Delicious. The difference between a tagging system like Flickr and Delicious, is that resources (Web sites) on Delicious can be tagged by multiple users. This is not the case with the resources (images) on Flickr.

We propose two adjusted methods (based on [3] and [4]) for clustering tags which address issues that are left open by the authors of [3] and [4]. So, in

total we have two 'types' of clustering techniques with two instances of each type. Thus, we implement 4 semantic clustering methods in order to address the second research objective.

For each semantic clustering technique, we use our own proposed method for dealing with the syntactic variations symptoms. This method deals with a lot of issues which are left open by other authors, e.g. [3] and [4]. This method addresses the first research objective.

We also propose two search methods, one for the non-hierarchical clusters, and one for the hierarchical clusters. After that, the results obtained from the implementations of these search methods are compared with the so called 'Dummy' search engine. This 'Dummy' search engine searches in the tag space, without using the knowledge about the semantic clusters or syntactic variation clusters. This comparison is also be made in the perspective of the different clustering techniques. It shows how the search methods perform on the two instances of each type of clustering technique. This gives insight into the possibilities of improving search and exploration in tag spaces.

We build a Web application, called XploreFlickr.com, which is accessible online [8]. The Web application makes it possible to compare the results from different clustering and search techniques directly on a subset of the Flickr database. This application addresses the third research objective.

1.2.1 Motivation

As we discuss in the related work section there has been a lot of research done for syntactic clustering and semantic clustering. Nevertheless, there is little or no literature where the focus is primarily on the improving of searching or exploring tag spaces. In this thesis we want to go one step further and empirically investigate searching in tag spaces with the knowledge of the derived syntactic and semantic clusters.

1.2.2 Scope

We consider only two types of clustering techniques: ones that produce non-hierarchical clusters, and techniques that produces hierarchical clusters. The hierarchical clusters can only be clusters of tags (relationships between tags).

It is also possible that you have a hierarchy of clusters (relationships between clusters), but we do not consider these type of hierarchies. Investigating these hierarchies of clusters might be interesting for future work.

Furthermore, the STCS framework uses only a relatively small part of the Flickr database. It does not run on 'live' Flickr data, this is for two reasons. The first reason is that the 'live' Flickr data set is too large to save one one hard disk and the Flickr API connection is too slow to do calculations on directly. The second reason is that it is not possible to perform clustering algorithms in a reasonable time on such a large data set as Flickr's.

1.3 Structure

This thesis contains six chapters. Every chapter starts with a small introduction and ends with a conclusion of that chapter. The thesis is organized as follows. Chapter 2 provides an overview of the related work. This chapter is divided in three sections, where each section addresses a part of the STCS framework, as described in Section 1.2. Chapter 3 contains a description of our design and the used methodology. This chapter starts with Section 3.1, describing the problem definition. Furthermore, Section 3.3 discusses the design and methodology of the three parts of the STCS framework. In Chapter 4 the development process of the STCS Framework is illustrated. Section 4.1 describes the gathering and preparation of the data set. The next three sections of that chapter discuss the implementation of the processes of identifying syntactic variations, deriving semantic clusters, and improving search and exploration in tag spaces, respectively. Chapter 5 gives an evaluation of all the parts of the STCS framework. In Section 5.1 the results of the first part of the STCS framework, removing the syntactic variations, are discussed. In Section 5.2 we discuss the results derived from the semantic clustering process. In Section 5.3 we discuss the results of the search and exploration improvement, based on the three aspects as stated in Section 3.1. Finally we present the conclusions from this chapter in Section 5.3.

Chapter 2

Related Work

In this chapter we discuss literature that addresses syntactic variations, semantic symptoms, and searching tag spaces. Before we dive into these three topics we analyze several similarity measures (measures that are used in either syntactic variation solutions or semantic symptoms solutions) as these are to be considered for the clustering methods. These similarity measures are discussed in Section 2.1. The literature which addresses syntactic variations, semantic symptoms, and searching tag spaces, is be analyzed in Section 2.2, Section 2.3, and Section 2.4 respectively.

2.1 Similarity Measures

In [9] the authors analyze several measures for tag similarity. Each measure is computed on data from Delicious [2] and a semantic grounding is provided by mapping pairs of similar tags in the folksonomy to pairs of synsets in Wordnet. The results expose some interesting features of the similarity measures. For detecting synonyms one should use tag or resource context similarities. For concept hierarchy, FolkRank and co-occurrence relatedness seemed to yield more general tags in their analysis. FolkRank gave the best results when considering tag recommendations.

In [10] the authors analyze the performance of two pattern matching techniques to identify syntactic variations in folksonomies, namely the Levenshtein distance and the Hamming distance. The Levenshtein distance between two

strings is given by the minimum number of operations needed to transform one string into the other. An operation is defined as inserting, deleting, or substituting a single character. For instance, the Levenshtein distance between 'wall' and 'balls' is 2, because two operations are needed. It can be considered as a generalization of the Hamming distance, which can only be used for strings of the same length and only considers substitution edits. So, the Hamming distance for 'tall' and 'bell' is 2, but the distance for 'tall' en 'bells' cannot be calculated.

For clustering of related tags several measures based on co-occurrence data are used in literature. In [3] the cosine similarity is used. The authors of [3] also tried different metrics to calculate the similarity between the pairs of vectors of co-occurrence data, including Euclidian and Manhattan distance, but achieved the best results with the cosine similarity measure. Metrics computing absolute distance like Euclidian and Manhattan showed to be inappropriate, since they are much more sensitive to significant variations in a few elements than little variations in a large number of elements. This is relevant when trying to find clusters of related tags.

2.2 Syntactic Variations

Syntactic variations between tags is a well-known symptom. Several authors have tried to deal with these variation by identifying them. As already mentioned, in [10] the authors analyze the performance of the Levenshtein distance and the Hamming distance. They perform an analysis over a large data set in two different ways, by (i) identifying pattern-candidate combinations, and (ii) the recognition of new tags (which are not a syntactic variation of others). In [10] 'pattern' is one of the tags in their data set, and 'candidate' is a syntactic variation of the pattern. The authors showed that both techniques provide similar results for some syntactic variation types, for example typographic errors and simple plurals/singulars. With identifying variations based on the insertion/deletion of characters, Levenshtein gets significantly better results than Hamming. However, both techniques do not perform as well as desired when identifying variations based in the transposition of adjacent characters (li-

brary/lirbary) or some kind of singulars/plurals (library/libraries). Moreover, both techniques improve their results ignoring candidate tags with length less than four.

In [3] the authors also use the Levenshtein similarity metric to group morphologically similar tags. They use a high threshold to determine 'similar' words (*cat* and *cats*) as well as misspellings (such as *theory* and *teory*). Within each group of similar tags, one is selected to be the representative of the group, and the occurrences of tags in that group are replaced by their representative.

2.3 Semantic Symptoms

In previous approaches, the semantic symptoms are addressed by either using a clustering technique which results in non-hierarchical clusters of tags, or a hierarchical graph of either tags or clusters of tags.

In [3] the authors present a complete framework where they address the syntactic variations in a tagging system, create clusters of semantically related tags, and within each cluster, identify the relationship between each tag pair. The semantic clustering algorithm of [3] distinguish itself, because tags can occur in multiple clusters.

For the clusters of related tags, the authors of [3] use the cosine similarity measure on the co-occurrence data. Given the highly similar pairs of tags, their algorithm considers each pair, for example, 'audio' and 'mp3', as seeds constituting an initial cluster, and then tries to enlarge this cluster by looking for tags that are similar to both initial tags. This procedure is recursively repeated for all tags, i.e., each new 'candidate' tag for a cluster must be similar to the whole (possibly enlarged) set of tags in that cluster. The algorithm generates a set of clusters, including a number of identical clusters, resulting from distinct seeds that are in fact similar to each other. It also generates highly similar clusters, differing in only a few tags, which are in many cases a consequence of the threshold to filter out unrelated pairs of tags. Two smoothing heuristics are used to avoid having a high number of these similar clusters. In order to determine the relationship between tags in a cluster, the authors use Swoogle [11] to find ontologies where both tags occur. They also make use of Wikipedia

[12], Google [13] and WordNet [14].

In [15] the authors create semantic clusters of tags by using co-occurrence data. For every tag in the data set they find the tags which co-occur the most with it. The authors use a cut-off value which is determined by the first and second derivative, where the co-occurrence count is on the y-axis and the tags are ordered descending on the x-axis). The tags above this cut-off value are placed in a graph with the co-occurrence counts as the weights of the edges. To split the clusters further, the authors use the spectral bisection algorithm [16]. Then they use the modularity function [17] to determine whether or not to reject or accept the partitioning. The algorithm then proceeds recursively on each accepted partition. The authors conclude that clustering techniques can and should be used in combination with tagging. They also suggest that these techniques can improve the search and exploration in tag spaces in general.

In [5], [6], and [4] a subsumption-based model is used to derive a hierarchy of semantically related tags. The title of [4] suggests that their methods result in an ontology induced from Flickr tags, but that is not the case. Their final result is a hierarchical representation of concepts. These hierarchies give no information about domains, ranges or the nature of the concept relationships. Thus, you do not have a real 'ontology'. The result of [5] and [6] are also concept hierarchies.

In [4] an adjusted subsumption model of [5] is used having as input co-occurrence statistics. They finally use this subsumption model for building hierarchical trees. The adjusted subsumption model consists of two steps. The first step is a pre-filtering step, where pictures with less than 2 tags, are filtered out as they have no meaning for co-occurrence statistics. The next step is the subsumption step. One tag subsumes another tag if the subsumption condition is fulfilled and if these two tags meet some statistical thresholds. The subsumption model from [4] differs from [5] as [4] uses additional conditions for statistical thresholds, like user count and tag count restrictions.

In [4] the models of [5], [6] are also implemented and tested on a snapshot of the Flickr database as of July 2005. This database consists of about 25 million images, which yield in 65 million annotations. The resulting trees is evaluated manually for all three models. The method proposed in [4] outperformed the

other two methods on the number of relevant relations, and on correctness.

Another example of a hierarchical taxonomy is proposed in [7]. The authors consider collaborative tagging systems. These systems have many casual users who annotate objects with free-form strings (tags) of their choice. Their algorithm builds a hierarchy of tags from annotation data. It is an extensible greedy algorithm that makes use of graph centrality. As input a similarity graph of tags is needed, this is a graph of all tags with the cosine similarity (based on co-occurrence) as the weight of the edges. The authors determined several features which impact the effectiveness of their algorithm. A prerequisite, as they propose, is that the data contains *natural hierarchical relations*. They argue that this seems to be a general feature of tagging data. An empirical study in [18] showed that a large proportion of tags in Delicious participate in hierarchical relationships.

2.4 Searching Tag Spaces

There is little literature where the focus is primarily on the improvement of search and exploration in tag spaces by using clustering methods. The discussed papers about semantic clustering state that the main goal the improvement of search and exploration in tag spaces is, but in fact, none of them investigates this aspect. The scope of the papers does not go beyond discussing the derived syntactic and/or semantic clusters.

In [19] and [20] seven different ranking algorithms for querying in tag spaces are discussed and evaluated. These algorithms can be applicable for users, tags and resources (or a combination of these). As we want to rank resources (Flickr images) we are only interested in ranking algorithms which are applicable for resources. Furthermore, there are some algorithms which are specially adjusted for GroupMe [21]. We also did not take these algorithms into account. Therefore, we only consider FolkRank [22], as SocialPageRank [23] is not suited for topic related ranking. Unfortunately, FolkRank is not applicable for the Flickr tagging system. FolkRank requires as the the number of users who annotated a certain resource. In Flickr this count is always 1, because only 1 user can upload a specific picture. This would impose a problem for the algorithm as

it is not designed for systems like Flickr. In other systems, like Delicious, it is possible to have multiple users linking to a specific resource (a Web page).

We could also have considered Latent Semantic Indexing for the ranking the images resulted from a search query, but we chose to use the average cosine similarity based on co-occurrence. The main goal of our research is not finding the best ranking method, but to investigate how semantic and syntactic clustering techniques can improve search and exploration in tag spaces.

Query relaxation can also be important when one searches tag spaces, but in the literature the focus is on querying directly in a database. Any extra knowledge, like you have with clusters, has not been taken into account.

2.5 Research Motivation

As previously mentioned, the proposed method in [10] had bad results for tags with a length smaller than 4. We want to improve the syntactic clustering part by addressing this issue. The authors also mentioned that both the Levenshtein and Hamming distance perform bad on tag pairs which are singulars and plurals of each other (e.g. 'libraries' and 'library'). Our method should also address this issue, together with other syntactic variations. The authors do not focus on improving search and exploration.

In [3] the authors propose a method that can derive an ontology from Flickr data. Their results look promising, but these results are derived manually. It is not yet possible to automatically achieve useful results. This is due to Swoogle, it has no good support for internally searching ontology documents. The method for removing syntactic variations is also limited as they use absolute Levenshtein measure with a high threshold to identify syntactic variations. This approach does not address the problem of syntactic variations between tags with length smaller than 4. Again, the goal is to improve this. The authors mention the aspect of improving search and exploration, but do not cover this.

The authors of [15] conclude that clustering techniques can and should be used in combination with tagging. The authors of [15] show that clusters can help with giving suggestions for related tags when searching or exploring tag spaces. The authors also suggest that clusters could be used for identifying

ambiguous tags (homonyms). This is a clear motivation for us to investigate clustering techniques further. Once more, the topic of improving search and exploration of tag spaces is not covered extensively by the authors.

In [5], [6] and [4] a subsumption-based model is used to derive a hierarchy of semantically related tags. The method of [4] outperformed the other two methods on the number of relevant relations and on correctness. However, the model proposed in [4] does not cover the symptoms of syntactic variations and typographical mistakes. The author of [4] proposes also to extend the model to be more robust, i.e., it should incorporate other aspects than only the co-occurrence count, when looking for parent-child relationships between tag pairs. We adjust the method proposed in [4] by using another approach for deriving the final hierarchical clusters, i.e., trees.

The reason why we have chosen to implement the method proposed in [3], is that their clustering algorithm allows tags to appear in multiple clusters. In this way homonyms could be detected and put in different contexts. We have chosen for [4], because his method is proven better than that of [5] and [6]. An alternative could have been the method in [7], his method is meant for *collaborative* tagging systems, like Delicious. The difference between a tagging system like Flickr and Delicious, is that resources on Delicious can be tagged by multiple users. This is not the case with Flickr.

2.6 Conclusions

As we have seen, there has been a lot of research done for syntactic clustering and semantic clustering. Nevertheless, there is little or no literature where the focus is primarily on the improvement of searching or exploring tag spaces. The discussed papers about semantic clustering all state as main goal the improvement of search and exploration in tag spaces, but in fact, none of them actually investigates this aspect. The scope of the papers does not go beyond discussing the derived syntactic and/or semantic clusters. In this thesis we want to go one step further and empirically investigate the searching in tag spaces with the knowledge of the derived syntactic and semantic clusters.

To summarize, for the syntactic variation recognition we use the normalized

Levenshtein distance in combination with the cosine similarity based on co-occurrence vectors. As motivated in Section 2.6 we use the adjusted methods proposed by the authors of [4] and [3] for semantic clustering. The method proposed in [4] is used to derive hierarchical clusters and the method of the authors of [3] is used for deriving non-hierarchical clusters.

Chapter 3

Framework Design

To answer the research questions, stated in Section 1.1, we propose the Semantic Tag Clustering Search framework (STCS). This framework consists of three parts. The first part deals with syntactic variations by clustering tags that are syntactic variations of each other and assigning a label to them. The second part of the framework addresses the problem of homonyms and identifying semantically related tags. The last, and final part of the STCS framework utilizes the clusters obtained from the first two parts to improve search and exploration in tag spaces.

In this chapter we first give the definition of the problem with which we are trying to deal with. After that, we discuss the STCS framework in detail in section 3.3. We finish this chapter with a short conclusion on the design and methodology.

3.1 Problem Definition

The input data set 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, and r is the ternary relationship $r \subseteq U \times T \times P$, defining the initial annotations of the users. We can split the problem definition into three parts: removing syntactic variations, finding semantically related tags and the improvement of searching in tag spaces with this knowledge.

3.1.1 Removing Syntactic Variations

The first goal is to remove any syntactic variations from tags. These syntactic variations could be the result of synonymous words like *'terrible'* and *'awful'*, but usually they are the consequence of typographic mistakes or syntactic/morphological variations. To detect syntactic variations between tags we need to create a set $T' \subset \mathcal{P}(T)$. Each element of T' represents a cluster of tags where each tag occurs only in one element (cluster), i.e., if $x, y \in T'$, and $a \in x$ and $b \in y$, then this implies $a \neq b$. Then we denote by m' the bijective function that indicates a label for each $x \in T'$, $m' : T' \rightarrow L$. Furthermore, for each $l \in L$ and some $x \in T'$, $l \in x$ exists, i.e., l is the label of cluster x.

To clarify this mathematical definition we give an example. Consider the set of tags (tag IDs) $T = \{1, 2, 3, 4, 5\}$. A possible T' could then be $T' = \{\{1, 3\}, \{2, 4\}, \{5\}\}$. The mappings could then be $\{1, 3\} \rightarrow \{1\}$, $\{2, 4\} \rightarrow \{4\}$, and $\{5\} \rightarrow \{5\}$. The set L then equals $\{1, 4, 5\}$, these tags are the labels.

3.1.2 Finding Semantically Related Tags

The second goal of this thesis is to create semantic clusters of tags. We want to group similar tags together, based on their semantic meaning. This means that we have to create a set T'' that clusters elements from $l \in L$. This denotes that we cluster only the tags that are labels of a syntactic cluster. An example of a semantic cluster is {new york, manhattan, hudson bridge, central park}. A tag should be able to be present in multiple clusters. This way, we can identify these tags as homonyms, i.e., they are related with multiple clusters and therefore have multiple meanings.

3.1.3 Improving Search and Exploration in Tag Spaces

We define the 'improvement of search and exploration in tag spaces' by the following aspects:

- The clusters provide information with which more and/or better results are retrieved;
- The search engine recognizes syntactic variations and homonyms;

- The precision of the search engine is increased.

3.2 Similarity Measures

In this section we discuss the different similarity measures used in the STCS framework. This section also introduces the notation and use of the different similarity measures.

3.2.1 Levenshtein Distance Measure

The Levenshtein distance is a metric for measuring the amount of difference between two strings (i.e., the so called edit distance). The Levenshtein distance between two strings is given by the minimum number of operations needed to transform one string into the other, where an operation is an insertion, deletion, or substitution of a single character. It can be considered a generalization of the Hamming distance, which is used for strings of the same length and only considers substitution edits. It is often used in applications that need to determine how similar, or different, two strings are.

For example, the Levenshtein distance between 'hair' and 'stairs' is 3, since the following three edits change one into the other, and there is no way to do it with fewer than three edits:

1. hair → shair (insert of 's' at the beginning)
2. shair → stair (substitution of 'h' for 't')
3. stair → stairs (insert 's' at the end).

From now on, we call this distance the *absolute* Levenshtein distance. We denote it by alv_{ij}, which is the absolute Levenshtein distance between tag i and j.

In the STCS framework we use the *normalized* Levenshtein distance, or also called the *normalized* Levenshtein similarity. We denote this by lv_{ij}, which is the normalized Levenshtein distance between tag i and j. The normalized Levenshtein value is defined as

$$lv_{ij} = 1 - \frac{alv_{ij}}{\max(\text{length}(t_i),\text{length}(t_j))} \ , \tag{3.1}$$

The normalized Levenshtein distances addresses the string lengths. If you have two strings, of both length 24, then an absolute Levenshtein distance of 3 is not large. However, with two strings of length 6 this distance is quite large (it is 50% of the tag length). According to the absolute Levenshtein distance these two distances are the same. But the normalized Levenshtein distances are in this case 0.875 and 0.5. This indicates that, according to the normalized Levenshtein distance, the two pairs of strings do not have the same distance, i.e., the first pair is more similar.

3.2.2 Co-occurrence Data and the Cosine Similarity

To measure the semantic relatedness between tags, we use the cosine similarity based on co-occurrence vectors. First, let us define what a co-occurrence matrix is, what the properties of it are, and how co-occurrence vectors are defined.

Given that there are m tags in the database of the data set, we can construct a matrix $X \in {\mathbb{N}_0}^{m \times m}$ where X_{ij} denotes the co-occurrence count between tag i and j. One should note here that $\mathbb{N}_0$ represents the collection of natural numbers *including* zero, i.e., $\mathbb{N}_0 = \{0, 1, 2, \ldots\}$.

A property of X is that the diagonal contains only zeros,

$$\sum_{i=1}^{m} x_{ii} = 0$$

Furthermore, x_j denotes the jth column of X. This is also called the co-occurrence vector of tag j. We define the function vector(j), which is used from now on to denote the co-occurrence vector for a given tag j.

Given two columns from matrix X called a and b, we can calculate the cosine between these two co-occurrence vectors

$$\cos(a, b) = \frac{a \cdot b}{\|a\| \, \|b\|} \tag{3.2}$$

where

$$a \cdot b = \sum_{i=1}^{m} a_i b_i = a_1 b_1 + a_2 b_2 + \cdots + a_m b_m$$

and

$$\|a\| = \sqrt{\sum_{i=1}^{m} a_i^2}$$

The range of the function $\cos(a, b)$ where $a, b \in \mathbb{R}^m$, is $[-1, 1]$. In this case the vectors $a, b \in {\mathbb{N}_0}^m$, it then turns out that the range equals to $[0, 1]$. We can interpret $\cos(a, b) = 0$ as 'no semantic relatedness', and $\cos(a, b) = 1$ as 'fully semantically related'.

3.3 STCS Framework

The STCS framework is divided in three parts, namely: removing syntactic variations from tags, creating clusters of semantically related tags, and finally utilizing this knowledge to improve the searching. In Section 3.3.1, we describe the process for removing syntactic variations from tags. The resulting output serves as input for the next process, i.e., semantic clustering. The removal of syntactic variations is necessary, because in the end, we want to have clusters that contain semantically related tags, and not syntactic variations of tags.

For the semantic clustering we compare hierarchical versus non-hierarchical clustering methods. For the non-hierarchical clustering types we implement the method proposed by [3] and an adaption of that algorithm. For the hierarchical clustering types we use the method proposed by [4] and also an adaption of that method, so in total we have 4 methods to compare. In Section 3.3.2 we discuss these four clustering techniques.

The STCS framework also consists of a search engine which utilizes the syntactic and semantic clusters to improve searching, this is the final part of the framework and it is discussed in Section 3.3.3. Figure 3.1 gives an overview of the STCS framework.

3.3.1 Removing Syntactic Variations from Tags

The algorithm for the syntactic variation clustering uses an undirected graph $G = (T, E)$ as input. The set T contains elements which represent a tag id, and E is the set of weighted edges (triples (t_i, t_j, w_{ij})) representing the similarities between tags. To calculate the weight w_{ij} one needs the normalized Levenshtein

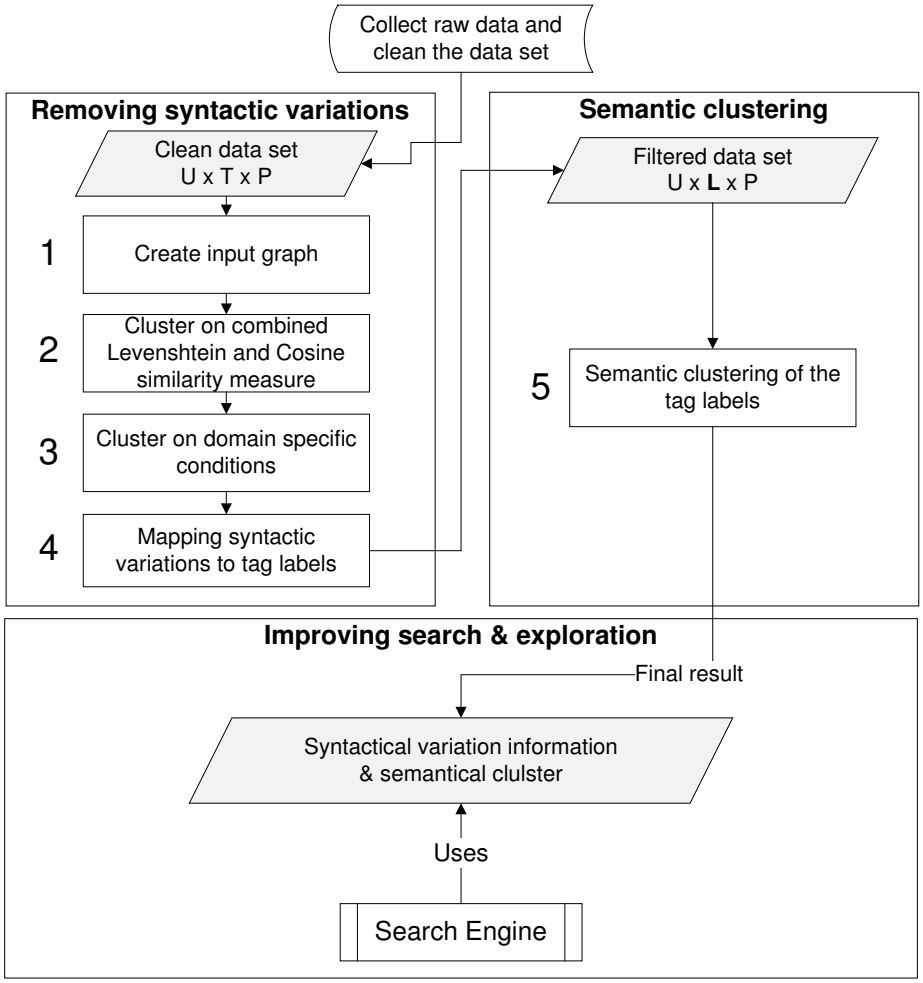


Figure 3.1: Overview of the STCS framework

distance lv_{ij} and the cosine similarity between tag i and j. The weight w_{ij} of an edge in the graph is then calculated as shown in equation 3.3.

$$w_{ij} = z_{ij} \times lv_{ij} + (1 - z_{ij}) \times \cos\left(\text{vector}\left(i\right), \text{vector}\left(j\right)\right) \ , \tag{3.3}$$

where

$$z_{ij} = \frac{\max(\text{length}(t_i), \text{length}(t_j))}{\max(\text{length}(t_k))} \in (0, 1] \text{, with } t_i, t_j, t_k \in T \ . \tag{3.4}$$

Normalized Levenshtein values are not representative for short tags, that is why the cosine value gets more weight as the maximum tag length gets shorter. This yields better results for shorter tags. Let us clarify this with an exam-

ple. Given two tags *'walk'* and *'wall'*, the normalized Levenshtein value equals $1 - 1/4 = 3/4$, a high value for words which are not syntactic variations.

Thus, if we use only the normalized Levenshtein distance in this case, these words would be marked wrongly as syntactic variations of each other. To address this problem we use the cosine similarity based on co-occurrence vectors and give it more weight for shorter tags. The cosine similarity indicates the level of semantic relatedness between two tags. The cosine similarity for *'walk'* and *'wall'* is so low that the framework correctly identifies that these words are not syntactic variations of each other.

To build the input graph we first make a list of triples t_i,t_j and w_{ij}, where $w_{ij} > \alpha \in (0, 1)$. The α indicates the condition for which we consider two tags as possible syntactic variations. When creating the list, only the pairs where $t_i < t_j$ should be considered as w_{ij} equals w_{ji}. With this list, the input graph can be build by creating nodes (tag id's) and edges that are on the list. After this process finishes, we create a root node and connect each 'cluster' of tags with this root node. The root is connected to a randomly chosen tag from each cluster. An example of an input graph for the syntactic clustering algorithm is shown in Figure 3.2. For example, $\{1, 5, 6, 7\}$ is a cluster which is connected to the root by the randomly chosen tag '1'. The root node functions as a pointer to clusters, which is used in the algorithm.

An overview of the algorithm for the syntactic clustering, which uses this input graph, is described in Algorithm 1 (this is step 2 of Figure 3.1).

Lines 1 and 2 indicate that the algorithm traverses through each cluster of the initial input graph. It checks every edge in a cluster, and if the weight of that edge is below a certain threshold β, the edge is cut. When an edge is cut, a

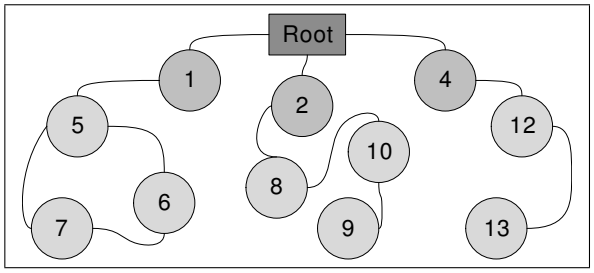


Figure 3.2: An example of an input graph for the syntactic variation clustering algorithm

Algorithm 1 Creating clusters that remove syntactic variation

Require: Input data assumed

- the set T of distinct tags and a root node *root*
- the set E of edges of the graph, contains triples (t_i, t_j, w_{ij}) where $t_i, t_j \in T$, and w_{ij} is the calculated similarity (as in Equation 3.3) between t_i and t_j
- β the threshold for the edge weight (similarity), if $w_{ij} < \beta$ then the edge is cut

```
1: for all t ∈ T∧ there exists a path from root to t do
2:   for all edges e = (t_i, t_j, w_ij) ∈ E that are part of the cluster of t do
3:     if w_ij < β then
4:       E = E − {(t_i, t_j, w_ij)}
5:       if no path from root to t_i then
6:         create link from root to t_i
7:       end if
8:       if no path from root to t_j then
9:         create link from root to t_j
10:      end if
11:    end if
12:  end for
13: end for
```

check is performed on the tags that are connected by that edge. For both tags we analyze if they are still connected to the root node. If that is not the case, an edge is added from the root to that tag (indicating that this tag now belongs to a new cluster). This process is described in lines 5 to 9.

The algorithm also ensures that a tag only appears once in a cluster, i.e., the root node does not point to two tags t_i and t_j such that there exists a path P from t_i to t_j with $root \notin P$. This is done by performing a depth traversal for each cluster after the algorithm finishes. When traversing we mark the nodes as 'visited'. If we encounter a node which has already been visited, we then know that the root is pointing to two identical clusters. The connection between the root and the current traversed cluster is cut. In this way the algorithm produces a set of distinct clusters where each tag can only appear once in a cluster. Let us clarify this with an example by look at the example input graph in Figure 3.2. Consider that we visited all nodes from the first cluster, i.e., the nodes $\{1, 5, 6, 7\}$. In the figure nodes 6 and 8 are not connected, but let us assume they are. If we now traverse through the second cluster we encounter

node 6. We known that we already visited this node and therefore remove the connection between the root and node 2. This action 'creates' a new cluster with nodes $\{1, 2, 5, 6, 7, 8, 9, 10\}$. The result of the algorithm is a graph that contains clusters of tags that are syntactic variations of each other.

Furthermore, we find that the data from Flickr has a domain specific property: there are a lot of camera and lens types present. Because different cameras and lenses are semantically related and the names often differ only by one or two characters, there are many different cameras and lenses in one cluster. To solve this, we propose a so called 'domain specific' step to deal with these domain specific properties. Step 3 of Figure 3.1 is this 'domain specific' step, which is similar to the previously described algorithm in step 2 of Figure 3.1, but with some slight modifications. The only thing that changes is line 3 of Algorithm 1). Here a new condition is used to replace the old condition for cutting an edge. For the Flickr Service we cut an edge if the extracted numbers from the tags are not equal to each other. For example, an edge connecting *'Canon EF 24-105mm f/4 L IS USM'* and *'Canon EF70-200mm f/4L IS USM'* is cut as '241054' does not equal '702004'. If only one of the two tags contains numbers, or the extracted numbers are equal, no edges are cut.

In step 4 of Figure 3.1 we create a new data set which contains the tags that are a label of a syntactic cluster, and the corresponding associations. The label of a cluster is the most frequently occurring tag in the data set. The newly created data set is used as input for the next steps. Table 3.1 shows an example of a cluster created at the end of the syntactic variations removal process (steps 1 to 4 of Figure 3.1).

Table 3.1: An example of some syntactic variations

Washington, D.C.
Washington
washington d.c.
washington dc
Washington, D.C.
Washington, DC
washingtondc

3.3.2 Semantic Clustering

As Figure 3.1 shows, the semantic cluster process starts when the syntactic variations have been removed from the data set. The algorithm we use to create the semantic clusters, step 5 of Figure 3.1, is described in Algorithm 2. This algorithm has originally been proposed by [3]. The algorithm is different from classical clustering algorithms. Instead of using the centroid, to calculate the distance between two clusters, all tags are used. This had the advantage that all the elements within a cluster must be similar amongst each other, instead of being similar just to the centroid. We improve the algorithm by replacing a heuristic for merging similar clusters by two new heuristics. In the final part of the research, we compare these two techniques ([3] and our adaption of that) and a third, hierarchical clustering technique.

Non-hierarchical Clustering

In lines 1 to 10 of 2 the initial clusters are created. This is done by starting with each tag as a cluster, and adding the rest of the tags to that cluster if they are sufficiently similar to that cluster. A tag is sufficiently similar if the average cosine of that tag with respect to all elements in the cluster are larger than χ. This is shown in line 5.

The set of initial clusters can contain many duplicate or nearly duplicate clusters. Therefore we need to merge some of the clusters, which is done in lines 11 to 20. In [3], two heuristics are proposed for this purpose. The first heuristic merges two clusters if one cluster contains the other. This means that if the larger cluster contains all the tags of the smaller, we remove the smaller cluster. The second heuristic checks if clusters differ within a small margin, i.e., the number of different tags in the smaller cluster represents less than a percentage of the number of tags in the smaller and larger clusters. If this is the case, then the distinct words from the smaller are added to the larger cluster and the smaller cluster is removed. The latter heuristic has limitations, because it uses a constant percentage, i.e. a constant threshold for merging clusters.

First, let us clarify these limitations in more detail. If one would use a constant threshold, it is hard to choose a threshold where the larger clusters do not merge to quickly and the smaller clusters too slowly. This is because

Algorithm 2 Semantic clustering

Require: T is a set of distinct tags, χ is the minimum average cosine with a cluster for the tag to be added to that cluster
Require: avgcosine (a, b) gives the average cosine between elements $(a - b)$ and b
Require: normdiff (x, y) gives the normalized difference between clusters x and y

$C = \{\emptyset\}$
for all $t \in T$ **do**
  $c = \{t\}$
  for all $t' \in T$ similar to t **do**
    if average cosine of t' with all tags in c is above χ **then**
      $c = c \cup \{t'\}$
    end if
  end for
  $C = C \cup \{c\}$
end for
$C' = \{\emptyset\}$
for all $y \in C$ in descending order of cluster size **do**
  for all $y' \in C$ in descending order of cluster size $\wedge\ y' \neq y$ **do**
    if $y' \subseteq y \vee$ avgcosine $(y', y) > \delta \vee$ normdiff $(y', y) < \varepsilon$ **then**
      $C = C - \{y'\}$
      $y = y \cup \{y'\}$
    end if
  end for
  $C' = C' \cup \{y\}$
end for

the maximum allowed number of different elements ($|D|$) is growing constantly with the size of the cluster. This is clear when we analyze the function which calculates the maximum number of different elements for the set to be merged. The function is: $f(|c|) = \lfloor \varepsilon \cdot |c| \rfloor$ where ε is the already mention threshold. For example, for $\varepsilon = 0.20$ we have $f(|c|) = \lfloor 0.20 \cdot |c| \rfloor$. If we evaluate $f(30)$ we get 6. This means if we have a cluster c with $|c| = 30$ and a cluster C with $|C| \geq 30$, c would be merged into C if $|D| \leq 6$ ($D = c - C$). This also means that any clusters with size below 4 is not merged, because $f(4) = 0$. To address this we propose a dynamic threshold. We find that a dynamic threshold, instead of a constant one, improves the clustering technique.

We employ the first heuristic that the authors proposed (which is a trivial one), but we do not use the second one as it has limitations due to the constant threshold. We replaced this second heuristic with two new heuristics, which we call the second and third heuristic from now on. The second heuristic consid-

ers the *semantic relatedness* of the difference between two clusters. The third heuristic considers the *size* of the difference between two clusters in combination with a dynamic threshold. We show that these three heuristics, the first proposed by [3] and the second and third proposed in this thesis, improve the clustering technique originally proposed by [3].

The three heuristics are used in line 14 of the algorithm introduced earlier. The part $y' \subseteq y$ represents the first heuristic, $\text{avgcosine}\,(y', y) > \delta$ represents the second heuristic and $\text{normdiff}\,(y', y) < \varepsilon$ represents the third heuristic.

Basically, the second heuristic, which we propose, merges two clusters C and c where $|C| \geq |c|$ when the average cosine avg of all $t \in \{c - C\}$ is above a certain threshold δ. The average cosine of these elements is defined as

$$avg = \sum_{d \in D} \frac{Avg_d}{|D|} \ , \tag{3.5}$$

where

$$Avg_d = \sum_{x \in C} \frac{\cos(\text{vector}(x), \text{vector}(d))}{|C|} \ , \tag{3.6}$$

and $D = c - C$, tag $d \in D$.

The third heuristic merges the clusters when the normalized difference between the clusters is smaller than a dynamic threshold ε. The normalized difference η is defined as

$$\eta = \frac{|D|}{|c|} \ , \tag{3.7}$$

where also $D = c - C$.

We propose to define threshold ε as

$$\varepsilon = \frac{\phi}{\sqrt{|c|}} \ , \tag{3.8}$$

and thus $f\,(|c|)$ can be described as

$$f\,(|c|) = \lfloor \varepsilon \cdot |c| \rfloor = \left\lfloor \phi \cdot \sqrt{|c|} \right\rfloor \ . \tag{3.9}$$

One is able to tune the distribution of maximum allowed difference between clusters by means of ϕ. The functional form of f is not linear, which would have been the case with a constant threshold. Thus, we can create a function that

suits the clustering process better.

Hierarchical Clustering

For hierarchial clustering we adapt, as already said, the algorithm of [4]. We first discuss the original proposed method and then the modification we propose.

The authors of [4] propose a subsumption model. In this model tag x potentially subsumes tag y (x is a parent of y) if

$$\begin{aligned} P(x|y) \geq t \text{ and } P(y|x) < t, \\ D_x \geq D_{\min}, D_y \geq D_{\min} \\ U_x \geq U_{\min}, U_y \geq U_{\min} \end{aligned} \tag{3.10}$$

where t is co-occurrence threshold, D_x is the number of documents in which tag x occurs, U_x is the number of users that use x in at least one image annotation. So the first step is to calculate the co-occurrence statistics.

Once the co-occurrence statistics are calculated, candidate term pairs are selected using the specified constraints. A graph of possible parent-child relationships is then built. To clean up the graph, the co-occurrence of nodes with ancestors that are logically above their parent are removed. So for example, for a given term x, and two potential parent terms p_i and p_j, if p_i is also a potential parent term of p_j, the p_i is removed from the list of potential parent terms for term x. At the same time, the co-occurrence of terms x, p_i and p_j in the given relationships indicates both that the (p_j, x) relationship is more likely than simple co-occurrence might indicate, and similarly that the (p_i, p_j) relationship should be reinforced. The author increments the co-occurrence statistic by 1 of each accordingly. After the paths are cleaned and reinforced, each leaf in the tree is considered and the 'best' path is chosen up to a root, given the (reinforced) co-occurrence weights. In the end, these paths are coalesced into trees. The best path is chosen by starting at a leaf and then choosing the best parent, i.e., the one with the highest co-occurrence.

We propose a modified version of this algorithm. Instead of choosing the 'best' path up to a root step-by-step, we use the longest path from a leaf up to a root. Figure 3.3 shows an example of a graph where we need to find the 'best'

path from leaf 'D'. With the step-by-step method proposed by [4], the 'best' path would be 'D,C,A', because $P(C|D) > P(B|D)$. With the longest path, the path would be 'D,B,A', because $P(B|D) + P(A|B) > P(C|D) + P(A|C)$. Clearly the path 'D,B,A' is preferred as the total co-occurrence values are larger, i.e., the total relationship is stronger.

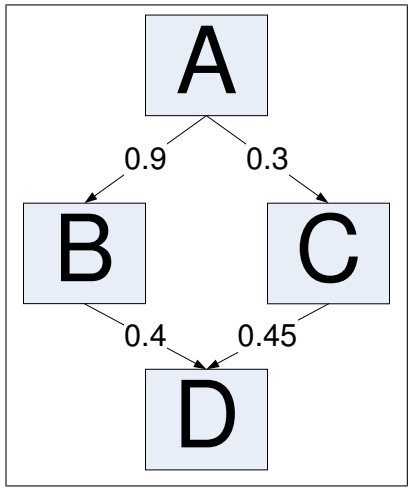


Figure 3.3: 'Step-by-Step' versus 'Longest Path', hierarchical clustering

3.3.3 Improving Search and Exploration in Tag Spaces

When the syntactic and semantic clusters are created, the search engine can utilize this information to improve searching. We now discuss each feature of the STCS search engine in detail.

Sorting the Results

The search engine sorts the pictures based on relevance with the query. The clusters are not used in any way to sort the pictures, instead the average cosine similarity is used. We use this measure, because it is more convenient and intuitive to use this measure, instead of using clusters to sort pictures. People do not search on a cluster, but on tag(s). As a results of this, the sorting of the pictures is the same for each cluster-driven search method. We can sort the results by defining a similarity measure between a query and a picture, and then sort the pictures based on this similarity.

We begin by defining the query q as a m dimensional column vector of tags q_i,

$$q = \begin{bmatrix} q_1 \\ \vdots \\ q_m \end{bmatrix}$$

We also define a picture p to be a n dimensional column vector of tags p_i,

$$p = \begin{bmatrix} p_1 \\ \vdots \\ p_n \end{bmatrix}$$

We can then define the function $f(q, p)$ which calculates the cosine between the query and the picture as in Equation 3.11.

$$f(q,p) = \frac{1}{n}\sum_{j=1}^{n}\left(\frac{1}{m}\sum_{i=1}^{m}\cos\left(q_i, p_j\right)\right) = \sum_{j=1}^{n}\left(\frac{1}{m \times n}\sum_{i=1}^{m}\cos\left(q_i, p_j\right)\right) \quad (3.11)$$

For a given query first all the results are collected. The syntax of the query determines which results should be returned, but we do not focus on this now as this is discussed in Section 4.4. For each picture the similarity $f(q, p)$ is calculated. The results are then sorted descending on the similarity and presented to the user.

Syntactic Variation Detection

An important feature of the search engine is the automatically replacement of syntactic variations by their labels. Steps 1 through 4 of Figure 3.1 generate tag labels which are mapped to tags. These tags are then called syntactic variations of their tag label. When a tag is not a syntactic variation it has just itself as the tag label. The search engine can utilize this information by searching for each keyword not only on the keyword, but also on all syntactic variations of the keyword. So when the user searches for 'self portraits', the system searches for 'self portraits', 'self-portraits', 'selfportraits' and even 'self portaits' (for example). This greatly increases the number of results, as showed in Section 5.3. This method is independent of the semantic clustering type (hierarchical

versus non-hierarchical).

Homonym Detection

Another feature of the search engine is that it is able to detect homonyms. If a word can have multiple meanings, the search engine asks the user for the right meaning (given the different contexts of the word). The results are returned according to the users answer. For this feature, the clustering technique is determining how the homonym detection takes place. This is because the information needed to detect a homonym is actually semantics about a keyword.

For the non-hierarchical clustering technique we utilize the number of clusters a tag occurs in. If a tag occurs in more than one cluster, it is considered as a homonym. The user then gets a message with the different clusters a tag is in.

For the hierarchical clustering technique we only look at tags which have at least one parent and at least one child in the tree (cluster), i.e., it must not be a root and also not a leaf in the tree. To determine whether or not this tag is a homonym we look at the average cosine (based on co-occurrence) between its children and its parent. If it is below a certain threshold λ, the tag is considered to be a homonym. The parent and the children are then the 'context' of this tag, and this information is presented to the user. An example of this is shown in Figure 3.4. Here the tag 'Apple' is said to be a homonym if the average cosine value of 'iPod' and 'iBook' with 'Fruit' is below λ.

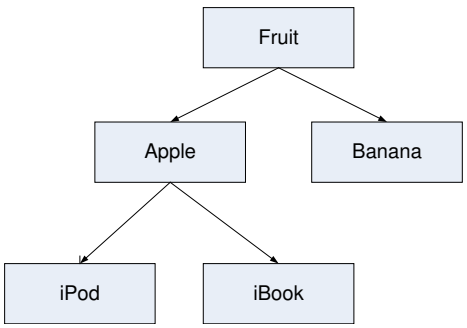


Figure 3.4: An example of a homonym in a tree

Providing Information about the Query

Searching is also improved by providing the user useful information about the query. Each clustering type provides different information, for instance, a hierarchical clustering technique can provide information concerning the hierarchy, and a non-hierarchical clustering technique can not. With this information the user understands the semantic structure of the query and the data, and can relax (or narrow) it manually depending on what the goal is.

We have chosen not to relax the queries automatically, because this is not the goal of this thesis. The goal is to find which clustering technique *helps the most* in improving searching. If we were to choose a clustering technique, this could have a biased effect on one or more of the clustering techniques. That is the reason why we choose to let the user relax the queries.

There are different scenarios we can think of. Sometimes the user does not find enough results. It could also be that there are too many results, or that the results are not specific enough (when searching on something specific). In any case, the user gets to see semantic information about the query. For the non-hierarchical clusters this means that the user sees the whole cluster for each tag. The tags within the cluster are ordered on cosine similarity with the tag so that the most similar tags are on top of the list. The hierarchical clusters give us the opportunity to tell the user more about each tag, as we can show the position of a tag within a tree. The user can then decide the relax the query more (by going one or more levels up) or to use a more constrained query (by going one or more levels down).

3.4 Conclusions

We have seen that the STCS framework consist of three parts, syntactic variation removal, semantic clustering, and improving search and exploration. The syntactic variation removal algorithm utilizes normalized Levenshtein distance combined with the cosine similarity based on tag co-occurrence data. Furthermore, we saw that semantic clustering is done by using two hierarchical and two non-hierarchical cluster techniques. Last, the search engine gives the user the possibility to relax and narrow the query, by providing the user semantic

information about the query. It also detects syntactic variations and homonyms in queries and sorts the results based on the average cosine similarity (based on co-occurrence data) between queries and images.

Chapter 4

Framework Implementation

In this chapter we start by looking at data processing, which is described in Section 4.1. After that, we discuss the implementation of the three parts of the STCS framework, syntactic clustering, semantic clustering and improving search and exploration. We close this chapter by a short conclusion.

Before we start, we should mention that the STCS framework is implemented in Java. For collections, like sets, stacks, queues and lists, we used Trove [24] in almost every case. This enabled us to have greater performance than the standard Java Developers Kit (JDK) implementation. There are some specific tools and libraries which we used, which are discussed in each appropriate section.

4.1 Data Processing

This section shows an overview of how we obtained our data set and how we performed the data cleaning. Section 4.1.1 describes the data collection and Section 4.1.2 shows how we cleaned the data set.

4.1.1 Data Collection

For our experiments we collected a data set from the Flickr database. Our data set contains $1,683,111$ associations ($U \times T \times P$). To lower the processing time, the data was collected in parallel from two non-overlapping intervals [2008-01-14, 2008-08-01] and [2008-08-12, 2009-02-28]. The total number of associations,

the number of (distinct) users, pictures and tags are shown in Table 4.1.

Table 4.1: The data set before and after applying filters

	# associations	# users	# pictures	# tags
initial data set	1,683,111	57,009	166,544	317,657
cleaned data set	1,231,818	50,986	147,132	27,401

4.1.2 Cleaning the Data Set

After collecting the data, we first perform some cleaning steps. As already mentioned in Chapter 5, users have no limitations when they add tags to pictures. Because of this, the pictures in our initial data set have many unusable tags. To address this problem we apply the following filters (in this order):

(1) **Remove tags with a tag length larger than 32 characters**
The data set contained tags that are whole sentences, we want solely individual tags 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

(3) **Remove tags which are complete sentences or enumerations of tags**
Some people use a tag to write a whole sentence, others use a tag to represent multiple tags (separated by a space)

(4) **Remove tags which occur in less than 6 different pictures**
We identified tags which occurred less than 6 times to be a statistical outlier ($Averagetaglength - 1.5 \times IQR$).

(5) **Remove associations containing pictures that have only one tag associated with them**
An association like this is not useful, because the tag does not co-occur with other tags in that picture. This implies that this association does not provide any information for a co-occurrence based measure (used later in this thesis).

After applying these filters we have a final data set which we use as input for our framework. The total numbers of (distinct) users, pictures, and tags after applying the previously discussed filters, are shown in Table 4.1. The syntactic clustering makes use of the full cleaned data set, the semantic clustering part and the 'improving search and exploration' part use the data set of the top 5000 most frequent tags (for performance reasons).

4.2 Syntactic Variations

For the syntactic variations part, we created a command-line application which implements the steps described in Section 3.3.3. The application creates mappings between tag labels and possible syntactic variations of tag labels.

To implement the syntactic clustering algorithm efficiently, we used Neo4j [25] to handle the graph manipulation. Neo4j is a graph database. It is an embedded, disk-based, fully transactional Java persistence engine that stores data structured in graphs. We chose this library because it has good features for traversing a graph, with easy customization possibilities.

4.3 Semantic Clustering

For the semantic clustering part, we implemented two separate applications, one for the non-hierarchical clustering and one for the hierarchical clustering. For the non-hierarchical clustering techniques we implemented a command-line application. For the hierarchical clustering, we used a more convenient graphical user interface application (GUI). A GUI is more appropriate here, because there our more parameters (thresholds) to adjust.

Both of the applications produce 'trees' (clusters), the only difference is that for the hierarchical clusters these trees are directed and for the non-hierarchical undirected.

The output of the applications is a GraphML [26] file and a text file, which can be read in a relational database. The GraphML file is a XML based format which is used to describe graphs.

We used the application yED [27] to read the GraphML [26] files, and vi-

sualize them. The yED application has many features, including auto layout algorithms which can be used to visualize graphs in a convenient way. The yED application is useful, as we could easily inspect the created clusters. Figure 4.1 shows a screenshot of the yED working environment.

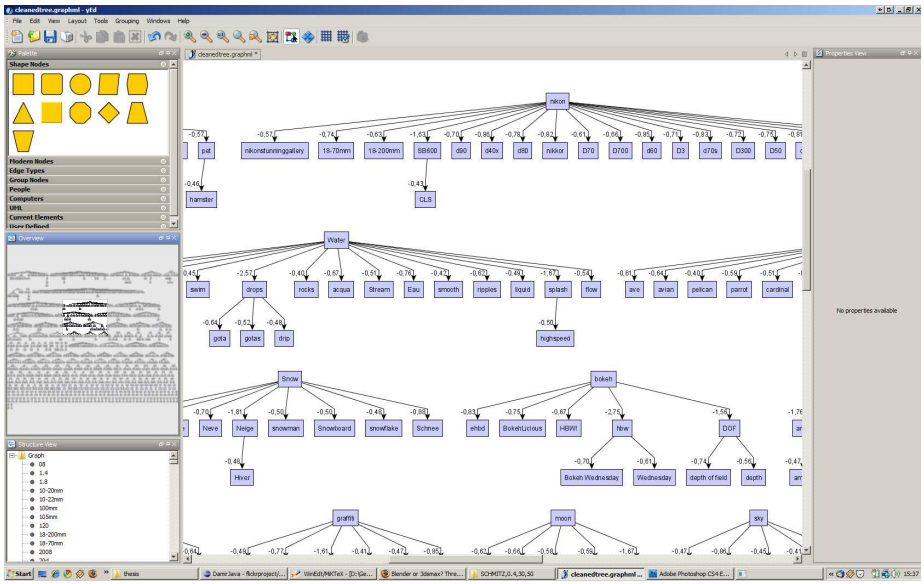


Figure 4.1: The work environment of yED

4.3.1 Non-hierarchical Clustering

We split the implementation of the non-hierarchical clustering process into two steps. First, a Neo4j graph database is created with the initial nodes/tags, this is the input graph as described in Section 3.3.1. Second, the created Neo4j database is taken as input, and the clustering process starts by cutting the edges (also described in Section 3.3.3). The final remaining Neo4j database is then used to print the mapping text file and the GraphML file.

4.3.2 Hierarchical Clustering

For the hierarchical clustering part, we created a graphical user interface for our application. Here the user can enter parameters and database information, as shown in Figure 4.2.

Instead of using the Neo4j graph database, we used an in-memory graph

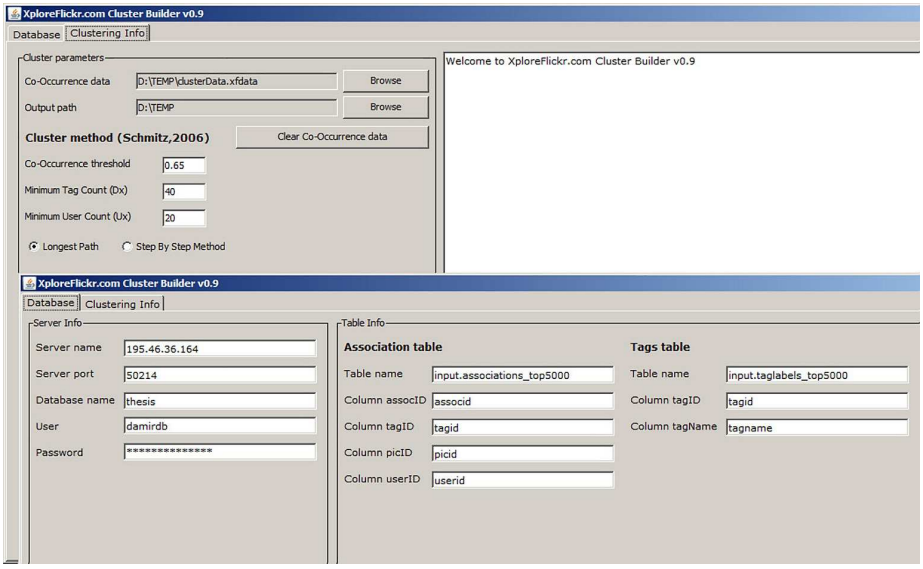


Figure 4.2: A screenshot of the Hierarchical Clustering Application

library called JGraphT [28]. The reason for this is that we needed quick access to the generated trees/clusters, therefore an in-memory solution seemed more feasible. Furthermore, the Neo4j graph database has great traversing possibilities, but not many implemented graph algorithms. JGraphT has a large number of graph algorithms implemented already, which are easy to use. We used, for example, the Bellman-Ford algorithm to find the longest path in trees. This is done by multiplying the edges by -1 and taking the shortest path. The result of this, with the Bellman-Ford algorithm, is that the longest path is found. This was our modification to the algorithm proposed by [4]. The output of this application is also a GraphML file together with a text file which can be imported in a relational database. The GraphML contains a collection of trees, to be more precise, directed acyclic trees.

4.4 Improving Search and Exploration

For this project, a search engine, called XploreFlickr.com [8], is developed. With XploreFlickr.com, the user can choose between different search methods, as described in Section 4.4.1. In Section 4.4.2 the architecture of XploreFlickr.com is explained.

4.4.1 The search methods

Figure 4.4 shows the search page of XploreFlickr.com. The user can enter a query (comma separated) and the search engine returns results according to a disjunction of the keywords in the query (OR query). If the user wants a tag to be present in the pictures, a '+' sign can be added in front of the keyword. So for example, the query 'taga, tagb, tagc' initially results in pictures which contain 'taga' or 'tagb' or 'tagc'. If the query is 'taga, tagb, +tagc', the results would have been ictures with ('taga' or 'tagb') and 'tagc'. Note that this '+' sign does not work for the Flickr and Dummy search methods.

We have also implemented an auto completer in XploreFlickr.com. When the user enters parts of some tag in xPloreFlickr.com, the auto completer suggests tag names from our data set. You can see an example of this in Figure 4.3. We make use of a JavaScript framework called Prototype [29] and an auto completer from [30].

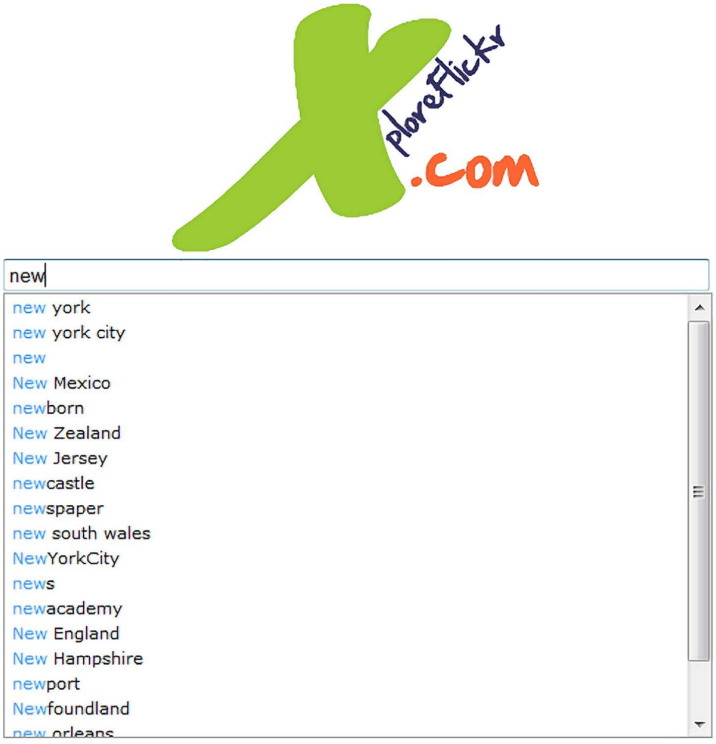


Figure 4.3: XploreFlickr.com: Auto completer

We can also see that the user can choose six different search methods. Ac-

tually these are four different search methods, but some with different datasets.

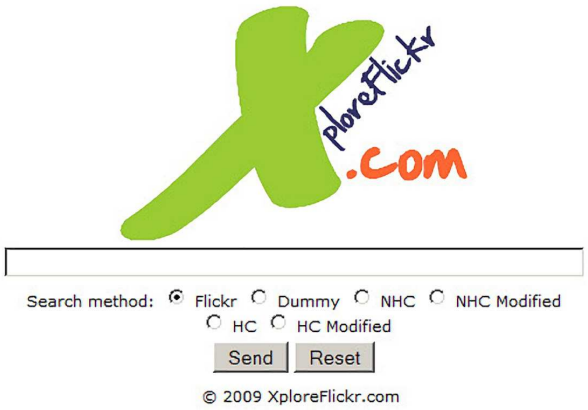


Figure 4.4: The XploreFlickr.com Web application

The labels 'Flickr' and 'Dummy' are search methods outside the STCS framework. The Flickr search method uses the Flickr API search method of the most interesting photos, it is not considered in this thesis as it searches the whole Flickr database. We implemented the Dummy searcher to simulate 'standard' search engines, it is searching for pictures in a trivial way, it only retrieves pictures which have exactly the tags the user has specified.

The labels 'NHC' and 'NHC STCS' represent the non-hierarchical clustering search method. The label 'NHC STCS' uses clusters created by the modified non-hierarchical clusterings algorithm.

The labels 'HC' and 'HC STCS' represent the search methods which are using hierarchical clusters. The label 'HC STCS' is different from 'HC' as it uses the clusters/trees which are created by using the longest path method, instead of the step-by-step method (see Section 3.3.2 'Hierarchical clustering').

The methods 'NHC', 'NHC STCS', 'HC' and 'HC STCS' initially all return the same images. We have chosen for this, as already explained in Section 3.3.3. The search engine of STCS lets the user relax the query if necessary, based on the information received of the search method. This information, of course, differs per search method.

Syntactic Variation Recognition

As already discussed in Section 3.3.3, we mapped syntactic variations to tag labels. These mappings are saved in a database table. When a user queries in XploreFlickr.com, then for each tag in the query, syntactic variations are searched, i.e., every tag in the query is replaced for the corresponding tag label.

Finally, when the search results are shown, the output is as in Figure 4.5 with respect to the syntactic variation detection. As a result of this syntactic variation detection, also pictures with only a misspelling or syntactic variation of a term are shown when somebody searches on the correct spelling of a specific term. For instance, in Figure 4.5 also pictures tagged only with 'self portait' or 'self portraits' are returned.

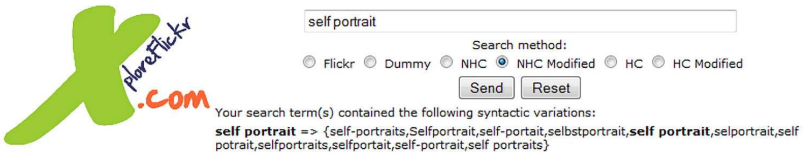


Figure 4.5: XploreFlickr.com: Syntactic variation detection

Homonym Recognition

In Section 3.3.3 we have explained how we detect homonyms in our clusters. This method differs per clustering type. We use different techniques for homonym detection in hierarchical clusters than for homonym detection in non-hierarchical clusters.

If a homonym is detected the user is asked to select one of the detected meanings of the homonym. After the selection the search results consist of images related to that specific meaning. For instance, when somebody searches on 'Apple' the user should be asked to select the meaning of 'Apple' by proposing two different clusters. One cluster related to 'fruit' and one cluster related to the 'brand'. An example of this is presented in Figure 4.6. The last column is not an actual cluster, but a choice not to select a meaning. After selecting one of the two clusters, only pictures which are tagged with 'apple', and tagged with at least one of the tags in the cluster, are shown in the search results. So, more concrete: if the cluster with *'Apple', 'iphone', 'iPod'* is chosen, only pictures

tagged with ‘Apple’ and (‘iPod‘ or ‘iPhone‘) are shown in the results. So, a picture must always be tagged with ‘Apple’ in combination with at least one other tag in the chosen cluster.

Figure 4.6: XploreFlickr.com: Homonym detection

Non-hierarchical Clustering

The non-hierarchical clustering search method is using a relational database for the cluster information and other information. We use a Microsoft SQL 2008 database that is stored on the same server as the Web application, as we discuss in Section 4.4.2.

Figure 4.7 shows an overview of the most important tables. An association links a user, a picture and a tag. Notice that the association table contains two tag identifiers, one for the tag label and one for the original, old, tag. We have chosen to replace, in the original associations table, all tags by their corresponding tag labels. This way, we can easily solve the problem of syntactic variations by converting the query to tag labels (replacing syntactic variations by their tag label), and then searching on the ‘`tagid`’ column. An example is shown in Table 4.2. Here tag 2 is identified as a syntactic variation of one or more tags, and is assigned to tag 6, which has now become a tag label. The same holds for tag 3, which is assigned to tag label 7.

For quick access to the syntactic variation mapping, we made a separate table called ‘`SYNTACTIC_VARIATIONS`’ to store them. We could have used the association table for this purpose (the same data is stored there), but we would need to query a much larger table each time a syntactic variation had to be

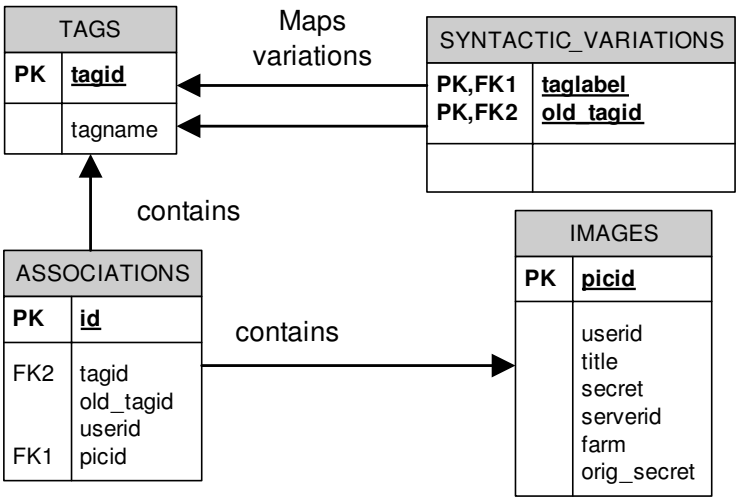


Figure 4.7: XploreFlickr.com: The tables in the database

Table 4.2: An example of the syntactic variations and tag labels

Old associations table				
id	tagid	userid	picid	
1	1	1	1	
2	2	2	2	
3	3	3	3	
4	4	4	4	

New associations table				
id	tagid	old_tagid	userid	picid
1	1	1	1	1
2	2	6	2	2
3	3	7	3	3
4	4	4	4	4

retrieved. Furthermore, the '`TAGS`' table stores the mapping between the tag ids and the tag names. The table '`IMAGES`' contains Flickr-specific information. Without this information we could not have rendered the pictures on the results page, the source URL is created by using a combination of this information. We decided not to store the URLs in the database as these might change when the Flickr decides to construct the URLs in a different way.

When the user submits a query, the search methods 'NHC' and 'NHC modified' return the pictures which meet the query. For each tag in the query the methods present their associated non-hierarchical clusters, if there are any for the tags in that query. This information is a textual representation. An example

of this is given in Figure 4.8.

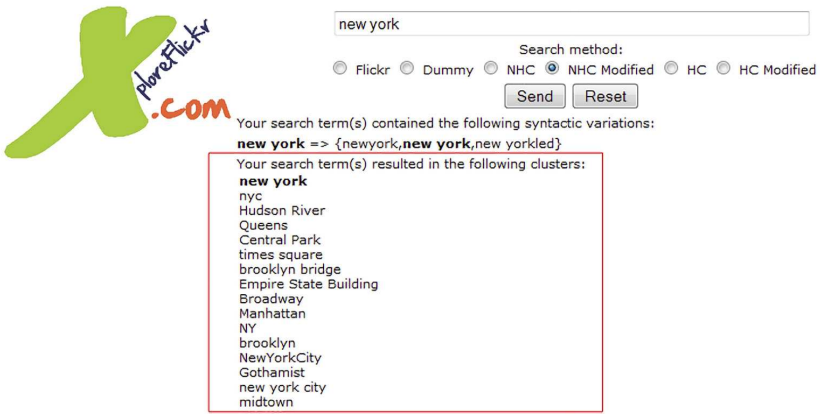


Figure 4.8: XploreFlickr.com: An example of a non-hierarchical cluster representation

Hierarchical Clustering

For the hierarchical clustering, we stored all the information about the clusters internally, i.e., in-memory. We actually implemented a service which operates on a JGraphT object. As the server starts, the nodes and edges of the trees are read. The service contains many helper methods, for example, finding a child of a tag, the parent of a tag, or the siblings of tag. For the syntactic variation detection the hierarchical clustering methods make also use of the table '`SYNTACTIC_VARIATIONS`'.

When the user submits a query, the search methods 'HC' and 'HC modified' also return the pictures which meet the query. For each tag in the query the methods return information about the tag in the tree. This information is a textual, hierarchical, representation of the tree where the actual tag is present. The actual tag is printed bold. An example of this is given in Figure 4.9

4.4.2 Architecture

XploreFlickr.com is running on a Tomcat [31] server. We use Spring [32] together with their 'Web Model View Controller' module to handle the Web application logic. With the Spring Web MVC we do not only benefit from the MVC pattern, but we also enjoy the power of dependency injection. Dependency injection is

Figure 4.9: XploreFlickr.com: A textual tree representation

a specific form of inversion of control where the concern being inverted is the process of obtaining the needed dependency. To give an example, all classes operate on interfaces, and not an actually objects. The objects are injected at runtime by the Spring container, the objects to be injected are specified in xml configuration files. This means that the classes are only dependent on interfaces and not implementations (real objects).

For both of the clustering techniques we needed to have fast access to cosine values between two tags. We have chosen to store this in-memory as the server starts up. There are approximately 12 million combinations which had to be stored. For this purpose, the Trove library is useful as the collections of Trove have a far more smaller memory footprint than the standard JDK implementations.

We modeled the search process for both of the clustering search methods according to the same flow of operations. When a user submits a query, the system follows this predefined flow of operations. The difference between each clustering search method is that each search method has its own implementation of these operations.

4.5 Conclusions

We have seen that several tools and libraries have been used to simplify the development of the STCS framework. For the syntactic variations and the non-

hierarchical semantic clustering, the most important library is the Neo4j library. The hierarchical semantic clustering benefited the most from the JGraphT library. Throughout the implementation the collections library Trove proved to be a valuable resource for high performance collections. Furthermore, the Web application development is greatly simplified by using Spring Web MVC.

Chapter 5

Evaluation

In this chapter we present and explain the results of removing syntactic variations, creating semantic clusters, and the improvement of search and exploration in tag spaces, i.e., we present and explain the complete STCS framework. In Section 5.1 we evaluate the detection of syntactic variation. In Section 5.2 we discuss the results obtained from our semantic clustering process. Section 5.3 evaluates our search results, based on the three aspects as stated in Section 3.1. Finally we present our conclusions from this chapter in Section 5.3.

5.1 Syntactic Variations

Table 5.1 shows we have $27,401$ unique tags, $50,986$ users, and $147,132$ pictures after cleaning the data. After applying the process of removing syntactic variations, as described in Section 3.3.1, we have $25,714$ tags left. Thus, we removed $1,687$ tags and replaced each of them with their corresponding tag label. We did not encounter any similar work in the literature where a larger data set is used.

To build the input graph for Algorithm 1, we calculate the normalized Levenshtein values for all the $27,401 \times 27,401$ combinations. For this, we employ a Java library called SimMetrics [33]. We choose $\alpha = 0.7$, because we assume that tags combinations with lower Levenshtein values than 0.7 are no syntactic variation of each other.

We apply a threshold value β of 0.62 (for cutting edges). This threshold

Table 5.1: Our data set obtained from Flickr: before and after applying syntactic clustering

	# associations	# users	# resources	# tags
cleaned data set	1,231,818	50,986	147,132	27,401
final data set	1,231,818	50,986	147,132	25,714

gives the best result on our test data set of 200 randomly chosen possible syntactic variations. Some examples of syntactic identical clusters are presented in Table 5.2. The tag on the top of every cluster, the base tag, is the most frequently occurring tag in our data set. The other tags in that cluster are replaced by their base tag.

Table 5.2: Some example clusters with syntactic variations

self portrait	**Canon EF 50mm f/1.8 II**	**Black and white**
Selfportrait	Canon EF 50mm f/1.8	blackandwhite
self-portrait	Canon 50mm f/1.8	Black & White
self portraits	EF 50mm f/1.8 II	black&white
selportrait	50mm f1.8 II	blackwhite
selfportait	canon 50mm f/1.8 II & black white	
selbstportrait	Canon EF 50mm f1.8 II	
self-portraits		
selfportraits		
self potrait		

Table 5.3 shows a cluster of 10 tags that is created when excluding step 3 of our framework. These 10 tags are actually part of a larger cluster (with a size of 72), but for simplicity we show only 10 random tags. Clearly this cluster is not correct as it groups several different type of lenses together. Table 5.4 shows what happens to four randomly chosen tags from Table 5.3 after including step 3 from our framework. The different type of lenses are now separated and grouped correctly.

In order to analyze the performance of our system, we define a test data set X that contains 200 randomly chosen tag combinations ($X \subset T \times T$) which the STCS framework classified as syntactic variations of each other. The distribution of the tag length for the test data set and the original data set is shown in Figure 5.1. This figure shows that the tag lengths approximately follow the same distribution.

The test data set X contains 313 distinct tags representing 139,277 associa-

Table 5.3: Wrongly classified syntactic variations, no domain specific step

Canon EF 70-200mm f/4 L IS USM
Canon EF 24-105mm f/4L IS USM
Canon EF 24-70 f/2.8L USM
Canon EF 200mm f/2.8L II USM
Canon EF 200mm f/1.8L USM
Canon EF 24mm f/1.4L USM
Canon EF 35mm f/1.4L USM
Canon EF 50mm f/1.8 II
Canon EF 85mm f/1.8 USM

Table 5.4: Three examples of correct clusters, created by using a domain specific step

Tag from old cluster	**Grouped with**
Canon EF 70-200mm f/4 L IS USM	Canon EF 70-200mm f/4L USM Canon EF70-200mm f/4L IS USM
Canon EF 24-105mm f/4L IS USM	CanonEF24105mmf4LISUSM Canon EF 24-105mm f/4 L IS USM
Canon EF 24-70 f/2.8L USM	Canon EF 24-70mm f/2.8 L USM Canon EF 24-70mm f2.8L USM Canon EF 24-70mm f/2.8L USM EF 24-70mm f/2.8L USM

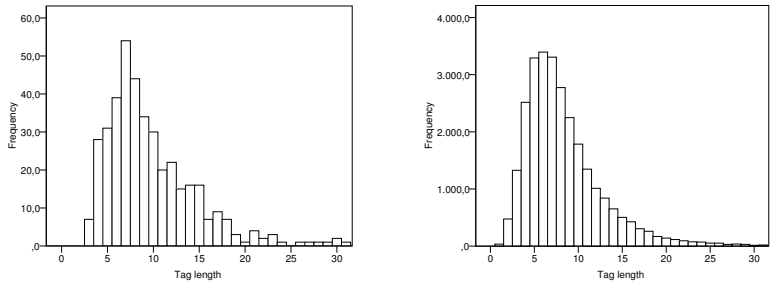
tions ($U \times T \times P$). We manually check all the 200 combinations in the test data set on correctness. The framework produces 10 mistakes. Thus, for this test data set, the precision error rate is 0.05. The 10 errors are shown in Table 5.5.

5.2 Semantic Clustering

In this section the non-hierarchical and hierarchical clustering approaches are evaluated. In Section 5.2.1 the results of the non-hierarchical approaches are discussed and in Section 5.2.2 we have a close look at the results of the hierarchical approaches.

5.2.1 Non-hierarchical Clustering

For the non-hierarchial clustering algorithm, we choose the threshold values which gave the best results when performing experiments. The threshold χ that determines whether or not a tag is added to a cluster during the initial



Test data set Original data set

Figure 5.1: Distributions of the tag length of our syntactic variations

Table 5.5: Syntactic clustering test data set, the 10 mistakes and 10 correct examples

10 mistakes		**10 correct examples**	
clouds	colours	lavender	lavendar
fars	pars	airbrush	airbrushed
hand embroidery	machine embroidery	Larsphotography	Larsphotography.com
clouds	Colores	autumn	automne
Western Australia	BestOfAustralia	paisaje	paisagem
original	originalart	turquoise	turqoise
Bangalore	mangalore	nature	natuur
clouds	color	full moon	fullmoon
blueberry	blackberries	Main Street USA	Main Street, USA
weed	seed	flat-coated retriever	flatcoatedretriever

cluster creation is set to 0.8. For the threshold δ we choose a value of 0.7. This threshold defines the minimum average cosine similarity when merging two sets when the smaller one has elements that the larger set does not have.

As parameters for the function that defines the dynamic threshold ε we use $\phi = 0.8$, as this gave the best results after experimenting. Figure 5.2 shows a plot of the threshold for $\phi = 0.8$.

Figure 5.3 and 5.4 illustrate the effect of using a dynamic threshold instead of a static one. One can notice that the maximum allowed different elements for the dynamic threshold is not a linear function. With the constant threshold you have more cluster sizes for which a merge never occurs than with a dynamic threshold.

Just like with the syntactic clustering process, we create a test data set to estimate the error rate. For this test data set, we randomly choose 100 clusters. The distribution of the sizes of these 100 random clusters and all clusters are

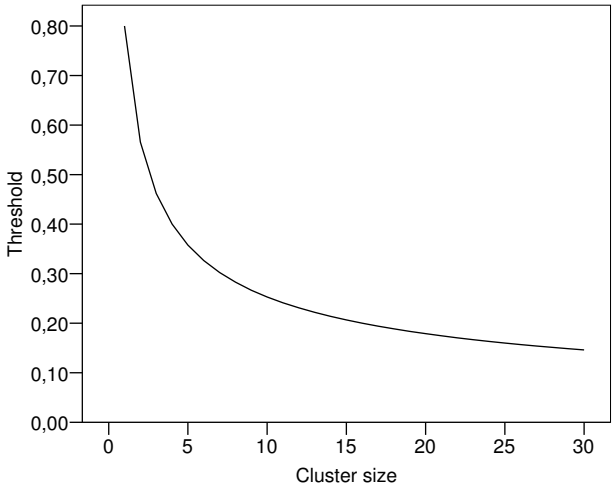


Figure 5.2: The threshold distribution for $\phi = 0.8$

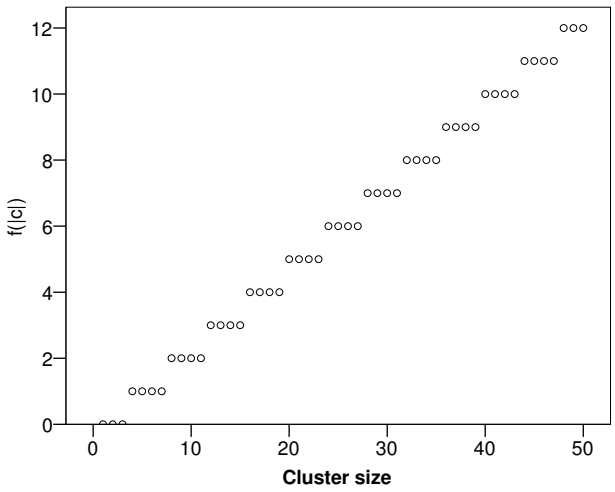


Figure 5.3: Maximum allowed different elements for a constant threshold, with $\varepsilon = 0.2$

approximately the same, as shown in Figure 5.5.

For each cluster we count the number of misplaced tags, i.e., elements that should be placed in another cluster. The total number of tags in this randomly selected test data set is 458 and we encounter 44 misplaced tags. Thus, for this data set, the error rate equals $44/458 \approx 0.096$.

One should note that most of the misplaced tags are part of a large cluster (size of 20 or larger). For example, a cluster of size 49 contained 21 misplaced

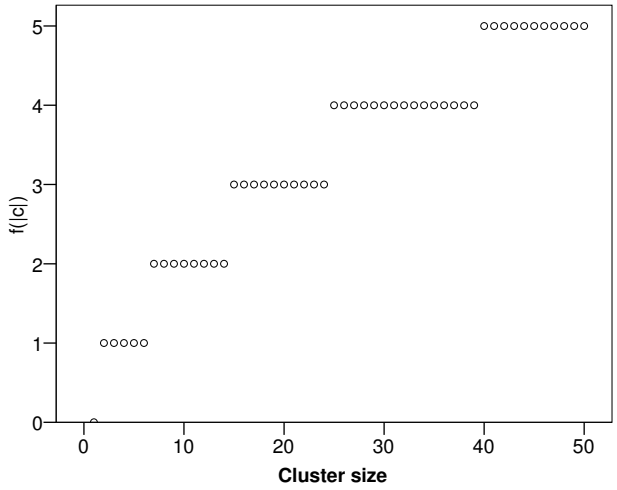
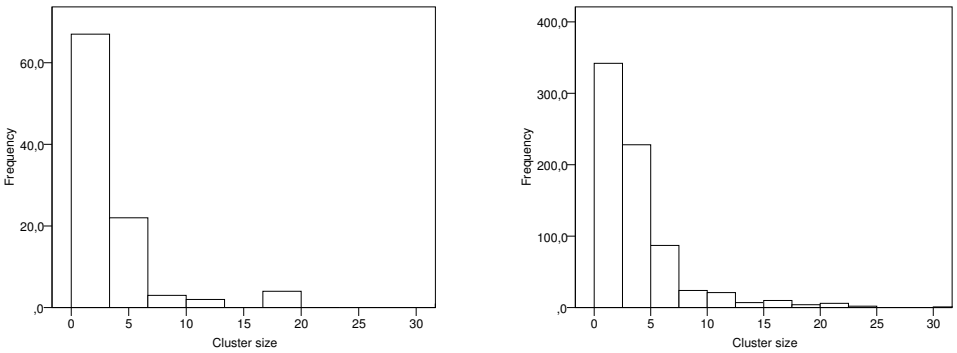


Figure 5.4: Maximum allowed different elements for a dynamic threshold ε, with $\phi = 0.8$



Test data set Complete data set

Figure 5.5: Non-hierarchical clustering: distribution of the cluster size, NHC Modified

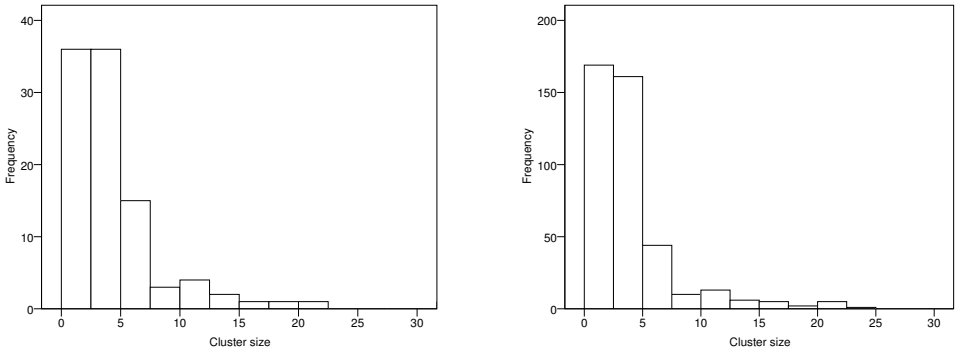
tags. This cluster contains tags about subjects like *'outdoor'* and *'nature'*. There are also, for example, tags which represents colors. Although colors do have something to do with *'nature'*, we mark these tags as misplaced as well. We encounter also one error in a small cluster: {jeans, feet}.

In general, the algorithm finds many relevant clusters. Examples are {rainy, Rain, wet, raining}, {turquoise, aqua, clear, cyan}, {iPod, iphone, mac} and {South, north, west}. Furthermore, a lot of clusters are found that actually contain tags from different languages. Examples of these clusters are {Praha,

Czech republic, praga, Czech}, {paris, frankreich, francia}, {Eau, Wasser} and {springtime, primavera}.

To benchmark our algorithm, we also implement the original, unadapted algorithm proposed by [3]. The difference between their approach and our approach is that they essentially use the heuristics 1 and 3 described in Section 3.3.2, with a constant threshold ε for heuristic 3. For the constant threshold, we find after conducting several experiments that the best results are achieved with a threshold of $\varepsilon = 0.2$.

Again, we create a test data set with 100 randomly chosen clusters to estimate the error rate. The distribution of the size of these clusters and the size of the clusters in the original data set is also shown in Figure 5.6. Both histograms show approximately the same distribution of cluster sizes.



Test data set Complete data set

Figure 5.6: Non-hierarchical clustering: distribution of the cluster size, Specia and Motta's algorithm

Once more, for each cluster we count the number of misplaced tags. The total number of tags in the test data set is 467. We encounter 61 misplaced tags, and thus, for this data set, the error rate equals $61/467 \approx 0.131$. When we compare this error rate to the error rate of our algorithm (9.6%), we conclude, based on the error rate on our test data set, that our algorithm outperforms the algorithm proposed in [3]. Furthermore we see that our algorithm produces 739 clusters, and the algorithm in [3] produces 421 clusters. Thus, our algorithm discovers more relationships between tags. The distribution of the cluster sizes is approximately the same for both methods. A performance summary is given in Table 5.6.

Table 5.6: Non-hierarchical semantic clustering, performance summary on the test data set

	Error rate	Number of clusters	Avg. cluster size	Min. cluster size	Max. cluster size
Specia & Motta	13.1%	421	4.6	2	63
STCS framework	9.6%	739	4.4	2	67

5.2.2 Hierarchical Clustering

To determine the thresholds (t, D_x and U_x) for the hierarchical semantic clustering, we performed several experiments. Table 5.7 shows the results of these experiments. The columns 'original' represent the original algorithm proposed by [4], which uses the step-by-step method to find a path from a leaf up to a root. The 'modified' column represents the adapted algorithm, which utilizes the longest path between a leaf and a root as criterion for path selection.

Table 5.7: Evaluation of the parameters for the hierarchical semantic clustering methods

Parameters			**# Tags**		**# Tags**		**# Tags**	
t	D_x	U_x	Original	Modified	Original	Modified	Original	Modified
0.4	30	30	1817	1825	367	362	1450	1463
0.4	30	50	1115	1115	255	254	860	861
0.4	50	35	1388	1393	298	295	1090	1098
0.4	100	30	792	795	192	192	600	603
0.45	30	30	1592	1601	358	354	1234	1247
0.65	30	15	960	961	268	269	692	692

We finally choose the parameters to be $t = 0.40$, $D_x = 30$ and $U_x = 30$. This gave the best trade-off between the number of correct edges and the error rate. Higher values than 0.40 for the parameter t resulted in the absence of valid subsumption pairs. We find that this is in contrast with the findings of [4], who reported an optimal t value of 0.80. For our data set, this value is too high. Although the author of [4] did not investigate the topic of searching tag spaces, the data set that is used for the semantic clustering is many times larger than our data set. This could be a factor which influences the optimal value of t. Figure 5.7 and 5.8 show us an example of what the difference is between 0.40 and 0.65 for the parameter t. We can see that with $t = 0.65$ we miss many useful subsumption pairs.

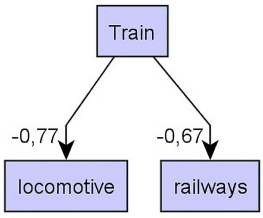


Figure 5.7: An example of a hierarchical cluster obtained with parameters $t = 0.65$, $D_x = 30$ and $U_x = 15$.

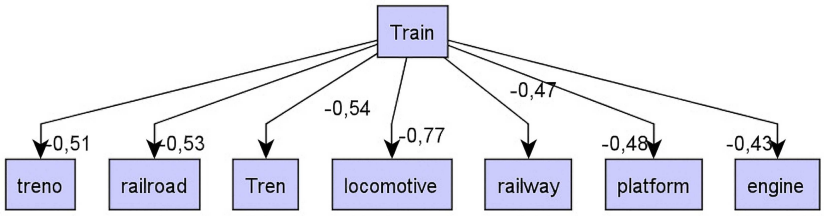
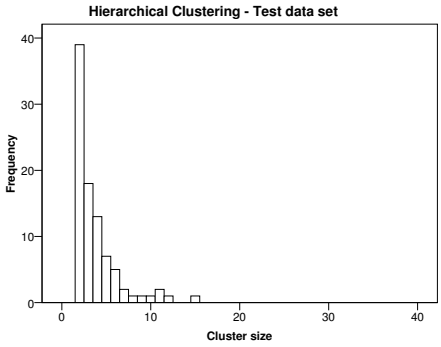


Figure 5.8: An example of a hierarchical cluster obtained with parameters $t = 0.40$, $D_x = 30$ and $U_x = 30$.

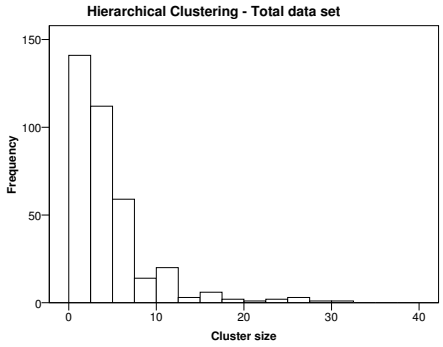
To evaluate both hierarchical clustering methods, we took 91 (roughly a third) clusters from the total data set of clusters. Figure 5.9 shows that the distribution of the cluster sizes, of the test data set and the total data set of the original algorithm, is approximately equal. Figure 5.10 shows the same comparison, but this time for the results obtained from our adaption to the algorithm of [4] (longest path instead of step-by-step). We can see that both figures show an equal distribution of the cluster sizes, thus, we can conclude that our test data set is representative with respect to the complete data set.

For each cluster, the proposed subsumption pairs, i.e., the edges, are evaluated. Each proposed subsumption pair is marked as correct, related, synonymous (including language variants), inverted, or noise (wholly erroneous). The edges that are marked as correct are truly of a 'type of' relationship. For example, Color → red is marked as correct, because red is a type of color. An other example of a correct edge, which is not strictly of the relationship type 'type of', is New York → Central Park. For generic terms like 'lake' and 'park', we considered instances of lakes or parks to be reasonable children.

An example of an edge that is of type 'related' is restaurant → food. The 'synonymous' relationship type is used when the parent and the child are two

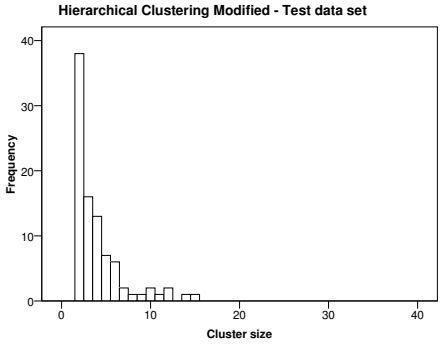


Test data set

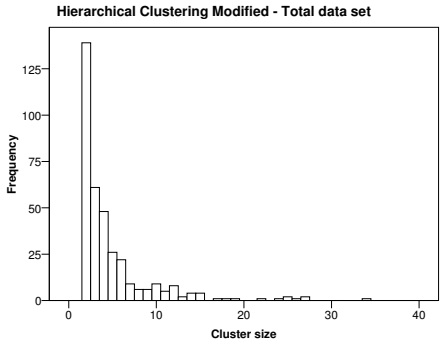


Complete data set

Figure 5.9: Distribution of the cluster size of original semantic clustering algorithm



Test data set



Complete data set

Figure 5.10: Distribution of the cluster size of the modified semantic clustering algorithm

synonyms, or syntactic variations, this can be also in different languages. An example of this type of edge is eyes → ojos and eyes → eye.

Inverted types of edges are edges where the actual child subsumes the actual parent. An example of this is red → color, this is clearly wrong as red is a color, and color is not 'a red'. Error or noise edges contain either Flickr specific or personal children or parents.

Table 5.8 shows what the evaluated results are for the original method proposed by [4], and our adaption to that (utilizing the longest path instead of the step-by-step method). We can see that the adapted method (with the longest path selection criteria) has a slightly better result on the total correct and synonymous relationships, i.e., more correct and less synonymous results. The

error rate is also approximately twice as low. Furthermore, the inverted rate is slightly higher for the adapted method.

Table 5.8: Relationship classification for hierarchical clustering

	Total #edges	Correct	Related / Aspect of	Synonymous	Inverted	Error / Noise
Original method	367	37.10%	19.35%	34.68%	3.23%	5.65%
Adapted method	360	39.70%	23.60%	30.34%	3.75%	2.62%

5.3 Searching Tag Spaces

In this section we evaluate the improvement of search and exploration in tag spaces. In Section 5.3.1 we compare the 'Dummy' search engine with the sorting algorithm of the cluster-driven search methods. Section 5.3.2 gives an overview of the results of the syntactic variation detection. In Section 5.3.3 we look at the results of the homonym detection and finally, in 5.3.4, we compare the extra information given by the cluster-driven search engines.

5.3.1 Sorting the Results

In this section we compare the cluster-driven search engines (NHC, NHC Modified, HC, and HC Modified) with the 'Dummy' search engine. The comparison is based on the 'precision' of the first 24 results when an user queries the system. Other statistical measures like 'accuracy', 'sensitivity', or 'specificity' are difficult to derive. For these measures you also need to know the 'false negatives' and 'true negatives'. This is difficult, because of the size of the data set. We should have evaluate every image in the data set for every query.

To evaluate our sorting algorithm, we randomly picked 300 tags from our data set. Subsequently we removed all the meaningless Flickr specific tags, like 'interstingness4', 'copyright 2008', and 'bigpicture2008'. It is not possible to evaluate the pictures when you query on terms like that. You do not know whether a picture is correct or incorrect. After the removal process we ended up with 107 tags. For all these 107 tags we entered a query in the 'Dummy' search engine and evaluated the shown results as correct (TP) or incorrect (FP). We repeated the process for one of the cluster-driven search engines.

We only considered the first 24 results for every query, because this is the number of results which is returned on the first results page. If a certain query resulted in less than 24 results, we only considered this number of results. To calculate the precision we used the formula as in equation 5.1.

$$precision = \frac{TP}{TP + FP} \tag{5.1}$$

Our obtained results are shown in Table 5.9. We performed an one-tailed Z-Test on our results to investigate whether our cluster-driven engines perform significantly better than the 'Dummy' search engine. Our null hypothesis and alternatives are defined as

$$u_1 : \text{ mean precision cluster-driven engines}$$

$$u_2 : \text{ mean precision 'Dummy' search engine}$$

$$H_0 \ : \ \mu_1 \leq \mu_2$$

$$H_A \ : \ \mu_1 > \mu_2$$

We used an α of 0.0001, so the critical value is $Z_{0.0001} \approx 3.72$. The Z value is 4.07. As $4.07 > Z_{0.0001}$ we reject H_0. So we conclude that our cluster-driven search engines perform significantly better than the 'Dummy' search engine with respect to the 'precision'.

Table 5.9: Obtained results from XploreFlickr.com, 'Dummy' vs cluster-driven search engine

	Precision	**Avg. # results**	**# Syntactic variations**
'Dummy' engine	0.73	968	0
cluster-driven engines	0.85	1112	29

Our test data set consists of queries of only one tag, because it is not possible to randomly select queries with a length larger than one tag. You cannot arbitrarily combine tags to construct a query. For instance, it makes no sense to combine the tags 'hamster' with 'self portrait'.

As shown in Table 5.9 the cluster-driven search engines perform significantly better than the 'Dummy' search engine. Therefore we conclude, considering our used search and sorting methods, that the cluster-driven engines also perform

better for queries containing more than one tag, when it comes to precision. We illustrate this with an example, consider that the results for querying 'Obama' and the results for querying 'election' are both better with the cluster-driven method than with the 'Dummy' search engine. It is then impossible to get better results with the 'Dummy' search engine when you would query on a combination of both tags.

5.3.2 Syntactic Variations

Table 5.9 shows that the number of results for the cluster-driven search engines is much larger than the number of results for the 'Dummy' search engine (for all the queries in the test data set). This is the result of the 'syntactic variation detection' feature of the cluster-driven engines. For queries with no syntactic variations the number of results is the same for both type of search engines. For queries containing syntactic variations, like in Figure 4.5, the number of results for the 'cluster-driven' engines are bigger than the results of the 'Dummy' search engine. In Table 5.10 you can see five examples of syntactic variations where the number of results is bigger for the cluster-driven engines in comparison with the 'Dummy' engine.

Table 5.10: XploreFlickr.com: examples of syntactic variation detection

	# Results	
Query	**'Dummy' engine**	**cluster-driven engines**
clouds	4889	5647
trees	2294	4622
Selfportrait	1378	3659
animals	540	1528
drops	553	996

5.3.3 Homonym Recognition

When considering the homonym detection, we make the distinction between the non-hierarchical and hierarchical clustering techniques.

Non-hierarchical Clustering

The method of [3] recognized 214 possible homonyms. Our adaption of that method in the STCS framework found 368 possible homonyms. We define as a

possible homonym as a tag which occurs in at least 2 different clusters.

After an analysis of all proposed homonyms, we found that both methods actually did not discover many homonyms. There are just a few correct homonyms. An example of this is the keyword ‘hot’. The system suggests, among others, the clusters ‘vacation’, ‘Holiday’, ‘hot’, ‘journey’, ‘tourism’, ‘trip’, ‘travel’, ‘place’ and ‘hot’, ‘fire’. This is shown in Figure 5.11.

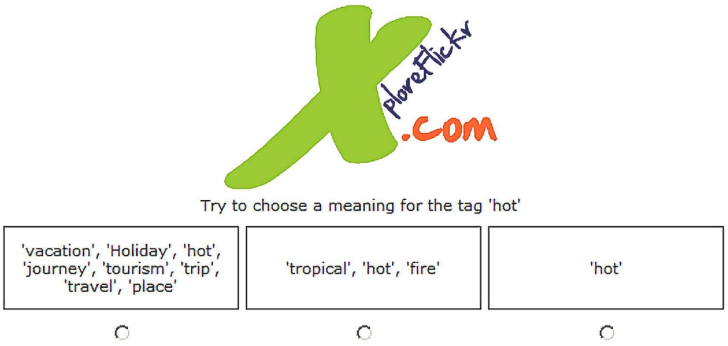


Figure 5.11: XploreFlickr.com: An example of a homonym

We have found that most of the time, the methods find ‘contexts’ for the tags. An example of this is the tag ‘beautiful’. Because this tag occurs in multiple different contexts, the user gets four large clusters presented where the context of ‘beautiful’ is determined. This is shown in Figure 5.12. Another example of a tag which can be put in different contexts, is shown in Figure 5.13.

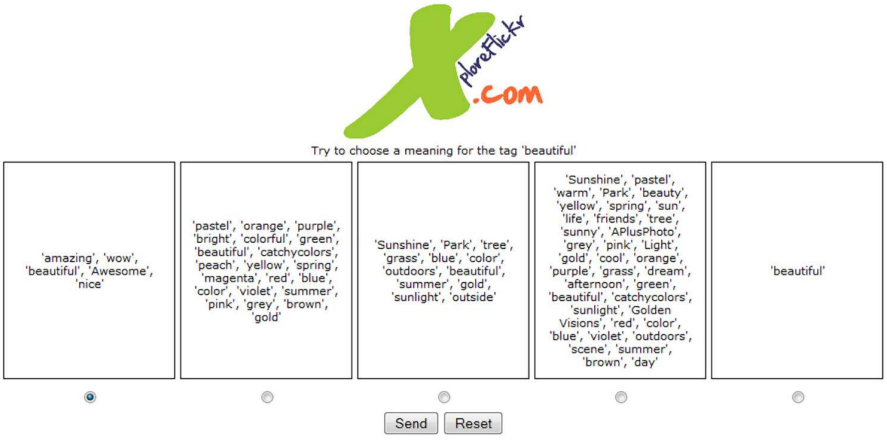


Figure 5.12: XploreFlickr.com: An example of different contexts: ‘beautiful’

Figure 5.13: XploreFlickr.com: An example of different contexts: 'daughter'

Table 5.11 shows the count of the found homonyms and contexts for the non-hierarchical clustering search methods.

Table 5.11: XploreFlickr.com: homonym recognition results for the non-hierarchical clusters

	# Possible homonyms	# Homonyms	# Different contexts
NHC	214	0	1
NHC Modified (STCS)	368	1	10

Hierarchical Clustering

For the hierarchical clustering, we found that it is not possible to find an appropriate value for λ, see Section 3.3.3 for more information on this threshold. If we choose the value to be $\lambda = 0.3$, no homonyms are suggested. If we choose $\lambda = 0.5$, then the system suggests many homonyms that are actually contexts, just as described above.

We performed the same analysis on the suggested homonyms for the hierarchical clusters. The results of this analysis, with $\lambda = 0.5$, are shown in Table 5.12. Both the HC and the HC Modified method have only 'contexts', instead of actual homonyms. The HC Modified search method does have slightly more contexts suggestions. Also the total number of possible homonyms is higher.

Two examples of tags which can be put in different contexts are 'pet' and 'blanket'. Figure 5.14 shows the suggested contexts for 'pet'. The system suggests a cluster with 'dog' and a cluster with 'hamster'. One can see this as two

different contexts for the tag ‘pet’. Figure 5.15 shows the same, but this time for the tag ‘blanket’. We see that the system suggests two contexts for ‘blanket’, namely two types of blankets.

Table 5.12: XploreFlickr.com: homonym recognition results for the hierarchical clusters

	# Possible homonyms	# Homonyms	# Different contexts
HC	46	0	16
HC Modified (STCS)	54	0	20

Figure 5.14: XploreFlickr.com: An example of different contexts: ‘Pet’

Figure 5.15: XploreFlickr.com: An example of different contexts: ‘Blanket’

5.3.4 Query Information

When you search on a regular tag search engine, you do not get any extra information about your query. Homonyms won’t be detected, syntactic variations

are not taken into account, related tags are not shown, and relationships between tags are not revealed. This is the case with our so called 'Dummy' search engine. Our cluster-driven search methods do have this features of information presentation about a query. In this section we evaluate whether this extra information improves the searching in a tag space as Flickr's.

To evaluate the extra information provided by the cluster-driven search engines, we picked the same 107 tags from our data set as in Section 5.3.1. For all these 107 tags we entered a query in all our cluster-driven search engines. Thus we performed $107 \times 4 = 428$ queries to get our results. Initially the results for all four methods are the same, as explained in Section 3.3.3 (except for the queries where homonyms are detected). To get different results for the four methods, we used the information retrieved from the clusters to narrow or relax the query. We explain this in more detail with an use case.

Consider we query on 'laugh'. In our data set, querying on 'laugh' results in pictures with laughing people, but also pictures as in Figure 5.16 are shown. This is not what you want. A more specific query might help. Therefore we

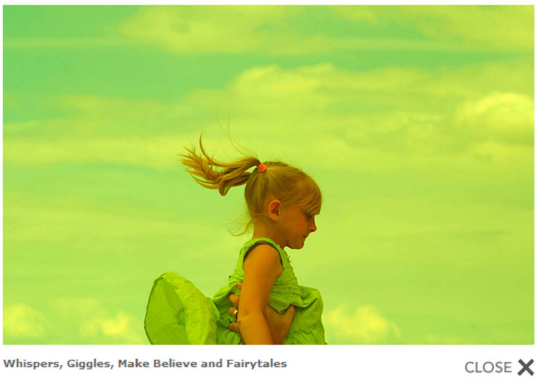


Figure 5.16: XploreFlickr.com: An example of a picture when querying on 'laugh'

can use the information gained from a cluster containing 'laugh'. In Figure 5.17 this cluster is shown. We adjust the query in 'laugh, smiling'. This improves our results, but we see in Figure 5.18 that another cluster is retrieved based on the tag 'smiling'. Thus, we add also a tag from that cluster to the query. Now our query looks like 'laugh, smiling, laughing' as shown in Figure 5.19. Analyzing the results, we conclude that adding the retrieved information from

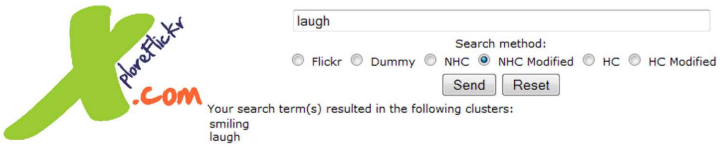


Figure 5.17: XploreFlickr.com: Querying on 'laugh'

Figure 5.18: XploreFlickr.com: Querying on 'laugh, smiling'

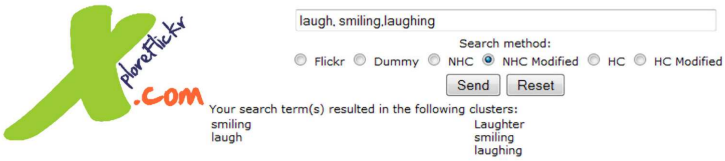


Figure 5.19: XploreFlickr.com: Querying on 'laugh, smiling, laughing'

the clusters resulted not only in more results (64 versus 134), but also in more precise results, considering the first page of the displayed results. For instance, we now see the picture from Figure 5.20 on the first page, where the results of the first query did not even contain this picture. Furthermore, the picture from Figure 5.16 is not on the first page anymore. We calculated the precision for all the 428 'optimized' queries. With optimizing the query we mean, making use of the cluster information as in the above example for the 'laugh' query. We tried to achieve a precision as high as possible for every query.

To investigate whether or not the relaxation of the queries improves the precision, we performed several z-tests. Our obtained results are shown in Table 5.13. To perform the z-tests we first define the precision averages as follows:

Figure 5.20: XploreFlickr.com: An example of a picture when querying on 'laugh, smiling, laughing'

u_1 : mean precision without relaxation

u_2 : mean precision NHC relaxation

u_3 : mean precision NHC Modified relaxation

u_4 : mean precision HC relaxation

u_5 : mean precision HC Modified relaxation

Table 5.13: XploreFlickr.com: Precision after query relaxation

	Average Precision	**Standard deviation**
Without relaxation	0.85	0.21
NHC	0.86	0.20
NHC Modified	0.88	0.20
HC	0.89	0.19
HC Modified	0.89	0.19

For each test we use an α of 0.05, so the critical value is $Z_{0.05} \approx 1.64$. The number of cases is for all samples equal to 107, i.e., $n = 107$.

First, we consider the non-hierarchical clustering method proposed by [3]. The null hypothesis and the alternative hypothesis is

$$H_0 \ : \ \mu_2 \leq \mu_1, H_A \ : \ \mu_2 > \mu_1$$

The Z value is approximately 0.43. As $0.43 < Z_{0.05}$, we accept H_0. So we conclude that the extra information provided by the original non-hierarchical clustering method does not significantly improve the precision, when compared to the precision without relaxation.

The modified non-hierarchial does improve the precision slightly more than the original non-hierarchical clustering method. The null hypothesis and the alternative hypothesis in this case is

$$H_0 \ : \ \mu_3 \leq \mu_1, H_A \ : \ \mu_3 > \mu_1$$

The Z value is more than twice as large, namely 1.19, but still note larger than the critical value, 1.65. Thus, we conclude that the extra information provided by the modified clustering algorithm also does not significantly improve the precision when compared to the precision of a non relaxing search method.

We have found that the hierarchical clustering techniques provide better information than the non-hierarchical clustering techniques. For the original hierarchical clustering algorithm, proposed by [4], the null hypothesis and the alternative hypothesis is

$$H_0 \ : \ \mu_4 \leq \mu_1, H_A \ : \ \mu_4 > \mu_1$$

The Z value equals 1.75, which is higher than the critical value of 1.64. We therefore conclude that the original hierarchical clustering algorithm does significantly improve the precision when compared to the precision without relaxation.

The same is true for the modified version of the algorithm (using longest path instead of the step-by-step approach), the Z value is even slightly higher,

namely 1.77. The corresponding null hypothesis and the alternative hypothesis is

$$H_0 \ : \ \mu_5 \leq \mu_1, H_A \ : \ \mu_5 > \mu_1$$

We can conclude that the hierarchical clustering techniques provide better information for improving precision on search results when user relax the queries.

Finally we can conclude that there is no significant different between the original proposed algorithm for hierarchical clustering and our adaption to that. A Z-Test with $\alpha = 0.05$ shows that the is no significant difference between the average precision rates of the two algorithms.

5.4 Conclusions

In this chapter, we have discussed several topics. The syntactic variations part of the STCS framework is evaluated by using a test data set. This test data set is manually checked for errors. We then calculated the precision error rate on this test data set, which equals 0.05.

This chapter also contains an evaluation of the semantic clustering. Here, we discussed the two non-hierarchical and the two hierarchical clustering techniques. For the non-hierarchical clusters we considered also the precision error in the clusters. We have seen that the precision error for the original non-hierarchical clustering is larger than the one for the STCS version of the same algorithm. For the hierarchical clustering techniques we evaluated a subset of the proposed subsumption pairs. We have found that the STCS version of the hierarchical clustering algorithm performs slightly better than the originally proposed algorithm by [3].

For the part of improving search and exploration in tag spaces, we evaluated 4 aspects. First, we looked at the sorting of the results. We showed that the cosine-driven sorting algorithm provides high precision rates when considering the 24 first pictures on the results page. The syntactic variation detection is also evaluated from the perspective of the user. Furthermore, we considered the homonym detection for each cluster-driven search method and evaluated each of these. Finally, we looked at the improvement of the precision rate when using the provided information from the system. We performed several Z-tests to test

for the statistical significance and found that the hierarchical clusters perform better than the non-hierarchical clusters.

Chapter 6

Conclusions and Future Work

In this chapter we draw conclusions, answer our research questions, and discuss future work. Section 6.1 answers the research questions and draws several other conclusions. In Section 6.2 we discuss the possibilities of future work.

6.1 Conclusions

In this thesis we present the STCS framework for dealing with syntactic variations, and deriving semantic clusters. We answer the research questions in the upcoming sections.

6.1.1 Syntactic Variations

The first research question,

> *How can one deal with syntactic variations and typographical mistakes?*

is answered by the algorithm for creating syntactic clusters. For the syntactic clustering process we propose a measure that uses the normalized Levenshtein value in combination with the cosine value based on co-occurrence vectors. The results show that domain specific tags can cause unwanted results. We expect

that this is also the case with tagging systems other than Flickr. The domain specific step in the STCS framework is therefore highly recommended in this context. Furthermore, abbreviations of words are not correctly clustered by the syntactic variation clustering algorithm. This is a result of our assumption that tag combinations are not syntactic variations of each other when the normalized Levenshtein value is smaller than α.

6.1.2 Semantic Clustering

The second research question,

> *How can one deal with semantic symptoms (homonyms, synonyms, and related tags) in tagging systems?*

is answered by the comparison of two hierarchical clustering techniques and two non-hierarchical clustering techniques. With these two types of clustering techniques, we can provide information to systems for them to detect homonyms, synonyms and related tags.

Non-hierarchical Clustering

The algorithm for the semantic non-hierarchical clustering process produces promising results. We find that the algorithm is able to find clusters of tags represented in different languages, besides the regularly semantic related tags. An example of such cluster is {Praha, Czech republic, praga, Czech}.

There are also some problems. Certain clusters are quite large. The result of this is that there are clusters which contain tags that are individually semantically related, but the semantic relatedness as a whole, is low. This is caused by a large context, for example *'outside'*. Of course, many things are outside, like *'trees'*, *'buildings'*, *'clouds'*, and *'sunny'*. However, as a whole, their semantic relatedness is low. Thus, this cluster is too large to consider it meaningful.

Nevertheless, on our test data set, the STCS version of the non-hierarchical clustering algorithm outperforms the original method, as proposed in [3], by using two new heuristics. The results obtained from our experiments show that our method performs better in terms of precision, and produces finer-grained clusters.

Homonyms are detected by looking at tags which appear in more than one cluster. If a tag appears in more than one cluster, it can be considered as a homonym. For the synonyms detection and related tags detection, the semantic clusters are used. Within a non-hierarchical cluster, one has no information about the structure or the type of the relationships between these tags.

Hierarchical Clustering

Like with the non-hierarchical clustering techniques, we implemented an existing hierarchical clustering technique and an adaption to that. When considering the precision increase of query relaxation with the knowledge that is provided by the clustering techniques, we found that there is no significant difference between the two implementations. We conclude that the adaption does not significantly improve the clustering algorithm.

We propose an algorithm to find homonyms in hierarchical trees of tags by looking at the average cosine similarity between the children and the parent of a tag. If the average cosine similarity is below a certain threshold, the tag can be considered to be a homonym. We find that this method does not find homonyms, but different contexts for tags.

Synonyms are discovered by using the children or parent of a tag. With a hierarchical clustering algorithm, one has the ability to navigate through a cluster. This way, users can relax queries by going one level up or constrain queries by going one level down in the tree. Related tags can then easily be found.

6.1.3 Improving Search and Exploration in Tag Spaces

The third, and last, research question,

> *What kind of clustering techniques are best to be used for improving search and exploration in tagging systems?*

is answered by the evaluation of the search information in Section 5.3.4. We conclude that querying, with information gained from hierarchical clusters, results in a significantly higher precision than querying without this information.

Querying with information gained from the non-hierarchial clusters did not lead to a significantly higher precision rate.

Also the syntactic variation recognition feature resulted in significantly better results with respect to precision and with respect to the number of results, in comparison to a regular tagging system search engine. Furthermore, we conclude that using the cosine similarity measure on co-occurrence data, to sort returned images when querying, results in better results with respect to precision.

6.2 Future Work

There are several aspects of the STCS framework which could be improved. First, we would like to improve the syntactic variation process. Our assumption that there is probably no syntactic variation between two tags when the normalized Levenshtein value is below a certain threshold, could be investigated further. For instance, *'New York City'* and *'NYC'* have a low normalized Levenshtein value, but in fact these tags are syntactic variations of each other. In general, one could research how to deal with syntactic variations in combination with abbreviations of words.

Furthermore, in this thesis we consider only two types of clustering techniques: ones that produce non-hierarchical clusters, and techniques that produces hierarchical clusters. The hierarchical clusters can only be clusters of tags (relationships between tags). It is also possible that you have a hierarchy of clusters (relationships between clusters), but we will not consider these type of hierarchies. Investigating these hierarchies of clusters might be interesting for future work.

The non-hierarchical semantic clustering leaves also room for improvement. We proposed in this thesis two new heuristics for merging similar clusters. The condition to merge was a disjunction of the two new heuristics and the earlier proposed heuristic. One could also consider other combinations, like a conjunction of the two new heuristics, to see if this improves the clustering process.

For the hierarchical semantic clustering process, one could consider another criterion than co-occurrence between two tags for the subsumption model. A

combination with cosine similarity and co-occurrence statistics would probably improve the proposed subsumption pairs.

For the improvement of search and exploration, by using clustering techniques, one could consider collecting, and analyzing real user statistics of users using different cluster-driven search engines. This in order to evaluate several search methods using different clustering techniques.

Bibliography

[1] Fake C., Butterfield, S.: Flickr - Online photo sharing service: `http://www.flickr.com/`.

[2] Schachter, J.: Delicious - social bookmarking: `http://delicious.com/`.

[3] Specia, L., Motta, E.: Integrating Folksonomies with the Semantic Web. In: 4th European Semantic Web Conference (ESWC 2007), Innsbruck, Springer (2007) 503–517

[4] Schmitz, P.: Inducing Ontology from Flickr Tags. In: 15th World Wide Web Conference (WWW 2006), Edinburgh, IW3C2 (2006) 206–209

[5] Sanderson, M., Croft, B.: Deriving concept hierarchies from text. In: 22nd ACM SIGIR conference on Research and development in information retrieval (SIGIR 1999), Berkeley, ACM Press (1999) 206–213

[6] Clough, P., Joho, H., Sanderson, M.: Automatically organising images using concept hierarchies. In: 28th ACM SIGIR conference on Research and development in information retrieval (SIGIR 2005), Salvador, ACM Press (2005)

[7] Heymann, P., Garcia-Molina, H.: Collaborative Creation of Communal Hierarchical Taxonomies in Social Tagging Systems. Technical Report 2006-10, Stanford InfoLab (2006) `http://ilpubs.stanford.edu:8090/775/`.

[8] van Dam, J.W.J., Vandic, D.: XploreFlickr.com: `http://www.xploreflickr.com/`.

[9] Cattuto, C., Benz, D., Hotho, A., Stumme, G.: Semantic Grounding of Tag Relatedness in Social Bookmarking Systems. In: 7th International Seman-

tic Web Conference (ISWC 2008), Karlsruhe, Vol. 5318, LNCS, Springer (2008) 615–631

[10] Echarte, F., Astrain, J.J., Córdoba, A., Villadangos, J.: Pattern Matching Techniques to Identify Syntactic Variations of Tags in Folksonomies. In: 1st World Summit on The Knowledge Society (WSKS 2008), Athens, Springer, (2008) 557–564

[11] Ding, L., e.a.: Swoogle: A search and metadata engine for the Semantic Web. In: 13th Conference on Information and Knowledge Management (CIKM 2004), Washington D.C. (2004) 652–659

[12] Sanger, L., Wales, J.: Wikipedia, the free encyclopedia that anyone can edit: `http://www.wikipedia.org`.

[13] Page, L. and Brin, S.: Google: `http://www.google.com`.

[14] Miller, G.A.: Princeton University: WordNet - a lexical database for the English language: `http://wordnet.princeton.edu`.

[15] Begelman, G., Keller, P., Smadja, F.: Automated Tag Clustering: Improving search and exploration in the tag space. 15th World Wide Web Conference (WWW 2006), Edinburgh (2006) 22–26

[16] Pothen, A., Simon, H.D., Liou, K.P.: Partitioning sparse matrices with eigenvectors of graphs. SIAM J. Matrix Anal. Appl. **11**(3) (1990) 430–452

[17] Newman, M.E.J., Girvan, M.: Finding and evaluating community structure in networks. Physical Review E **69** (2004)

[18] Kome, S.H.: Hierarchical subject relationships in folksonomies. Master's thesis, University of North Carolina at Chapel Hill (2005)

[19] Abel, F., Henze, N., Krause, D.: Ranking in folksonomy systems: can context help? In: CIKM '08: Proceeding of the 17th ACM conference on Information and knowledge management, New York, NY, USA, ACM (2008) 1429–1430

[20] Abel, F., Henze, N., Krause, D.: Analyzing Ranking Algorithms in Folksonomy Systems. Technical report, L3S Research Center (2008) `http://groupme.org/papers/techreport-ranking-in-folksonomies.pdf`.

[21] Abel, F.: Welcome to GroupMe! - The Social Semantic Web: http://groupme.org/.

[22] Hotho, A., Jäschke, R., Schmitz, C., Stumme, G.: Folkrank: A ranking algorithm for folksonomies. In: Proc. FGIR 2006. (2006)

[23] Bao, S., Xue, G., Wu, X., Yu, Y., Fei, B., Su, Z.: Optimizing web search using social annotations. In: WWW '07: Proceedings of the 16th international conference on World Wide Web, New York, NY, USA, ACM (2007) 501–510

[24] Friedman, E.D.: GNU Trove - High performance collections for Java: http://trove4j.sourceforge.net/.

[25] Eifrem, E.: Neo4j - The graph database: http://neo4j.org/.

[26] GraphML - file format for graphs: http://graphml.graphdrawing.org/.

[27] yWorks.: yED - powerful graph editor: http://www.yworks.com/en/products_yed_about.html.

[28] Naveh, B.: JGraphT - free Java graph library: http://jgrapht.sourceforge.net/.

[29] Stephenson, S.: Prototype JavaScript Framework: http://www.prototypejs.org/.

[30] Kirda, T.: Ajax AutoComplete for Prototype: http://www.devbridge.com/projects/autocomplete/.

[31] Apache Tomcat: http://tomcat.apache.org/.

[32] Johnson, R.: Spring - platform for Java enterprise applications: http://www.springsource.org/.

[33] Chapman, S.: SimMetrics package: http://sourceforge.net/projects/simmetrics/.