Incremental Cosine Computations for Search and Exploration of Tag Spaces

Raymond Vermaas, Damir Vandic, and Flavius Frasincar

Erasmus University Rotterdam
PO Box 1738, NL-3000 DR, Rotterdam, the Netherlands
info@raymondvermaas.nl, {vandic, frasincar}@ese.eur.nl

Abstract. Tags are often used to describe user-generated content on the Web. However, the available Web applications are not incrementally dealing with new tag information, which negatively influences their scalability. Since the cosine similarity between tags represented as co-occurrence vectors is an important aspect of these frameworks, we propose two approaches for an incremental computation of cosine similarities. The first approach recalculates the cosine similarity for new tag pairs and existing tag pairs of which the co-occurrences has changed. The second approach computes the cosine similarity between two tags by reusing, if available, the previous cosine similarity between these tags. Both approaches compute the same cosine values that would have been obtained when a complete recalculation of the cosine similarities is performed. The performed experiments show that our proposed approaches are between 1.2 and 23 times faster than a complete recalculation, depending on the number of co-occurrence changes and new tags.

1 Introduction

User-based content is becoming increasingly available on the Web. This content is often annotated using tags and then uploaded on social sites, like the photo sharing service Flickr. Because users can choose any tag they like, there is a large amount of unstructured tag data available on the Web. The unstructured nature of these tags makes it hard to find content using current search methods, which are based on lexical matching. For example, if a user searches for "Apple", (s)he could be looking for the fruit or for the company that makes the iPod.

There are several approaches available that aim to solve the previously identified problem [2, 4, 9–11]. In this paper, we focus on the Semantic Tag Clustering Search (STCS) framework [4, 9, 11]. The STCS framework utilizes two types of clustering techniques that allow for easier search and exploration of tag spaces. First, syntactic clustering is performed by using a graph clustering algorithm that employs the Levenstein distance measure in order to compute the dissimilarity between tags. As result of syntactic clustering, e.g., terms like "waterfal", "waterfall", and "waterfalls" are clustered. This means that when a user searches for one of these terms, all the terms that are syntactically associated will show up in the results. Second, semantic tag clustering is performed, where the aim is

to cluster tags that are semantically related to each other, e.g., "tree", "plant", and "bush". The STCS semantic clustering algorithm makes use of the cosine similarity measure, applied on tag co-occurrence vectors. Due to the similarities between the STCS framework and the other clustering approaches that rely on the cosine similarity, the results presented in this paper can be easily applied on these approaches as well.

An important issue with the STCS framework is that it cannot be applied in an incremental way. This negatively influences the scalability of the approach. In other words, if new pictures are added, all steps in the framework have to be executed again, which can take a significant amount of time when handling large amounts of data. Another reason for updating the cosines incrementally is that when a small number of new tags are added, most of the new calculations will yield the same result as the previous computations. This is a natural consequence of the fact that the tag co-occurrence matrix remains the same for a large number of tags.

In this paper, we investigate how the cosine similarity computation can be done incrementally, for the purpose of scaling semantic clustering algorithms in tag spaces. We consider two approaches for incrementally computing the cosine similarity. The first approach only recalculates the cosine similarities of tag pairs that are affected by a change in the tag co-occurrence matrix. The second approach computes the cosine similarities by reusing the previously computed cosines. The second approach refines the first approach as only the cosine similarities of the tags affected by the changes in the tag co-occurrence matrix are computed. For the evaluation of the proposed approaches, we use a reference approach that computes all cosine similarities in a given data set. The evaluation is based on a simulated process in a photo sharing Web application (e.g., Flickr.com), where the system receives a set of new pictures that need to be processed. These new pictures can be annotated using existing (known) tags but also with new (unknown) tags. For the evaluation, we measure the execution time of each approach and compare that to the approach that performs a complete recalculation of all cosines. We use a data set from the photo sharing site Flickr.com to perform the evaluation. The data set has been collected by Li [8] and contains 3.5 million pictures and 580,000 tags, uploaded in the period 2005-2008. We use a subset of 50,000 pictures for the initial data set and between 2,500 and 75,000 pictures as the sets of new pictures that are pushed through the hypothetical photo sharing Web application. The reason for using a subset of the complete data set of 3.5 million pictures is that the computation of all cosines for the baseline takes too long for large data sets.

2 Related work

There are a small number of approaches available in the literature that address the incremental (exact) computation of cosine similarities. One such approach is proposed by Jung and Kim [7]. The authors develop several methods for the incremental computation of similarity measures, including the cosine similarity,

in the context of merging clusters. The aim is to merge two clusters without calculating all similarities between the new cluster and all the other clusters. To achieve this, the geometrical properties of the cosine similarity are used to calculate the similarity of the new merged cluster using the previously computed similarities. The results of this approach showed the usefulness of this solution with respect to achieving a significant speed-up and also a good accuracy. Unfortunately, we cannot use this approach for our purposes as we consider the incremental computation of cosines irrespective of the clustering method that is being used.

Another approach to incremental cosine computation is proposed by Friedman et al [5]. The authors develop three techniques for incremental cosines: crisp, fuzzy, and local fuzzy. In the crisp cosine computation, the cosines are calculated using only the dot-product. Every time a new vector is added, the cosine similarities with all cluster centroids are calculated. In the fuzzy cosine clustering technique, a degree of cluster membership is assigned to each new vector. This process takes the size of the new vector into account. The last technique that is proposed by the authors is the local fuzzy-based cosine, which is a modified version of the fuzzy cosine computation. This technique takes the difference between small vectors and a large centroid into account when assigning multiple degrees of membership. We do not consider this approach as it is focused on clustering scalability issues and not on the incremental computation of cosine similarities in general.

Literature in the field of incremental clustering shows more approaches that might be related to our work. The evolutionary clustering technique proposed in [3] considers a trade-off between low history cost (similarity with a previous clustering iteration) and high quality of each iteration. This is achieved by considering the similarity between the old clusters and new clusters (history cost), and the cosine similarity between the newly added elements into account. The authors only tried their technique on k-means clustering and agglomerative clustering, but suggest that their approach should also work for other clustering techniques. We do not consider this approach as it also does not distinguish between the cluster-based computations and the cosine similarity computations.

Locality sensitive hashing (LSH), as proposed by Giones et al [6], can also be used to perform incremental cosine computations. The idea behind LSH is that elements that are close to each other in a high dimensional space have a high probability of having a similar hash. The process of locality sensitive hashing consists two parts: a pre-processing part and a querying part. In the pre-processing part, the hashes are calculated for all the elements in the data set. Next, all the hashes are transformed to their bitwise representation and the similarity between the hashes is measured using the Hamming distance. Subsequently, all the elements with a similar hash are clustered in buckets using the k-nearest neighbour method. In the querying part, the hash for the query is calculated and a bucket containing similar elements is returned. LSH has proven to have sub-linear query time and a small error compared to other approaches that aim to solve similarity problems. We do not consider LSH in our work as it

is an approximate technique and we aim to develop an exact technique for the incremental computation of cosines.

3 Incremental cosine computation design

In this section, we describe the different approaches that we propose for the incremental computation of cosine similarities. First, we discuss the details of the cosine similarity and the operations that should be taken into account when calculating the cosine similarity incrementally. Then, we describe two approaches for incremental cosine computations, the recalculation approach and the delta cosine approach.

3.1 Cosine similarity

The cosine similarity measure is used to measure the similarity between two vectors. This done by measuring the cosine of the angle between the two vectors. The cosine similarity ranges between 0 and 1 if the values in the vectors are positive. A cosine similarity of 1 represents complete similarity between the two vectors (i.e., the vectors point in the same direction) and a cosine of 0 represents complete dissimilarity (i.e., the vectors point in orthogonal directions). The cosine similarity is defined as:

$$\cos(\mathbf{a}, \mathbf{b}) = \frac{\mathbf{a} \cdot \mathbf{b}}{||\mathbf{a}|| \times ||\mathbf{b}||} \tag{1}$$

where $||\mathbf{a}||$ and $||\mathbf{b}||$ are the Euclidean norm of vectors $\mathbf{a}$ and $\mathbf{b}$, respectively, and $\mathbf{a} \cdot \mathbf{b}$ is the dot product between the vectors $\mathbf{a}$ and $\mathbf{b}$. The cosine similarity is calculated by dividing the dot product between the two vectors by the product of the Euclidean norm for both vectors, as shown in Equation 1. The dot product calculates the similarity between the vectors and the product of the Euclidean norms is used as a normalization factor.

In the STCS framework [4, 9, 11], the tag co-occurrence matrix is a central concept. This matrix contains for every tag pair the co-occurrence (i.e., how often two tags co-occur on pictures in the data set). A tag is represented as a vector of its co-occurrences with all other tags. This vector also includes a tag's co-occurrence with itself, which is by convention set to zero. The number of cosine computations that have to be computed between tag vectors grows quadratically with the number of tags in the data set. For n tags we have:

$$\text{Number of cosines to be calculated} = \frac{n^2 - n}{2} \tag{2}$$

This quadratic growth is a large bottleneck for the scalability of approaches that utilize the cosine similarity. Incremental cosine similarity computations are a possible solution to this problem, since there is no need to calculate the cosine similarity for every tag pair when new tags are added.

3.2 Approach 1: Incremental recalculation approach

The first approach is based on the incremental recalculation of cosines. By considering the changed tag co-occurrences, this method determines which cosines need to be recalculated. Incremental cosines are based on two types of update operations: an update of existing tag co-occurrences or the addition of new tags.

The update operation of already existing tags is quite simple. For every tag pair (i,j) that is updated (every tag has an index assigned), all the cosines for tag pairs of which tag i or tag j are part, need to recalculated. This is due to the fact that the cosine similarity uses the whole tag co-occurrence vector, rather than individual co-occurrences. As result of this, for every tag co-occurrence update of already existing tags, $2 \times n - 2$ cosines need to be updated (as 2 vectors have changed). Let us consider the co-occurrence matrix presented in Table 1. An example of an update on this matrix is shown in Table 2. In this example, the co-occurrence for the tag pair (3,4) is changed (from 2 to 3). This change causes the cosine similarities for tag pairs of which tag 3 or 4 are part to be recalculated. The "-" in Table 1 and 2 are co-occurrences of a tag with itself, which we consider to be zero by default.

Table 1. The original co-occurrence matrix, which contains 5 tags.

Tag	1	2	3	4	5	6
1	-	2	1	5	2	0
2	2	-	7	1	1	0
3	1	7	-	2	0	2
4	5	1	2	-	1	0
5	2	1	0	1	-	6
6	0	0	2	0	6	-

Table 2. The gray cells are the cosines of tag combinations that need to be recalculated after the co-occurrence between tag 3 and 4 changes.

Tag	1	2	3	4	5	6
1	-	2	1	5	2	0
2	2	-	7	1	1	0
3	1	7	-	3	0	2
4	5	1	3	-	1	0
5	2	1	0	1	-	6
6	0	0	2	0	6	-

The addition of new tags is slightly more complicated, since it consists of two sub-operations. First, all cosine similarities between the new tag and all the other tags need to be calculated. An example is shown in Table 3, where tag 7 is being added as a new tag. Second, all combinations between existing tags that have a non-zero co-occurrence with the new tag need to recalculated. This has to be done in order to ensure that also the new tags are considered in the cosine similarities of vectors involving old tags. For example, it could happen that a certain existing tag pair had a similarity of 0 before the new tag was added, but now has a similarity larger than 0, because of the newly introduced co-occurrences (from the new tag). A visual example of this operation is shown in Table 4. Note that although many cells are gray, which indicates where recalculation is needed, only half of the cosines are recalculated on account of the symmetrical properties of the tag co-occurrence matrix. After both operations are performed, we have a list of unique tag combinations for which the cosine similarity needs to be recalculated.

Table 3. Tag 7 is added to the tag co-occurrence matrix, new tag combinations are shown in gray.

Tag	1	2	3	4	5	6	7
1	-	2	1	5	2	0	1
2	2	-	7	1	1	0	0
3	1	7	-	3	0	2	6
4	5	1	3	-	1	0	1
5	2	1	0	1	-	6	0
6	0	0	2	0	6	-	0
7	1	0	6	1	0	0	-

Table 4. Gray indicates the existing combinations that need to be recalculated because tag 7 is added.

Tag	1	2	3	4	5	6	7
1	-	2	1	5	2	0	1
2	2	-	7	1	1	0	0
3	1	7	-	3	0	2	6
4	5	1	3	-	1	0	1
5	2	1	0	1	-	6	0
6	0	0	2	0	6	-	0
7	1	0	6	1	0	0	-

3.3 Approach 2: Delta cosine approach

The second approach first calculates the dot-product and the Euclidean distance for each incremental operation and then, when all the incremental operations are done, calculates the final cosine similarity. It uses both the changed and new co-occurrences in an iteration to determine for which tag pairs the cosine similarity needs to be recalculated, similar to the first proposed approach.

In contrast to the first approach, the two incremental operations are split in the second approach. First, the update operation for the existing tags is executed. During this operation the change to the dot-product for all affected cosines is considered for each update to the co-occurrences of the existing tags. In this step we consider the change to the $n \times n$ tag co-occurrence matrix A. The changes to the tag co-occurrences and the new co-occurrences are defined in a $m \times m$ matrix ΔA. The sizes of these matrices are such that $n \leq m$ and $m - n$ is the number of new tags. The set U_a contains indices of tag vector a for which the values are updated. Equation 3 is used to calculate the change to the dot-product of tag a and every other unchanged tag b ($a \neq b$).

$$(\Delta \mathbf{t}_a + \mathbf{t}_a) \cdot \mathbf{t}_b = \mathbf{t}_a \cdot \mathbf{t}_b + \sum_{i \in U_a} \Delta \mathbf{t}_{ai} \times \mathbf{t}_{bi} \tag{3}$$

Tags $\mathbf{t}_a$ and $\mathbf{t}_b$ are both existing tag vectors of matrix A. This approach lets us update only effected elements of the dot-product of the cosine rather than the whole dot-product, which addresses the scalability of this method. In case both tags $\mathbf{t}_a$ and $\mathbf{t}_b$ change, the updated dot product formula becomes:

$$\begin{aligned}(\Delta \mathbf{t}_a + \mathbf{t}_a) \cdot (\Delta \mathbf{t}_b + \mathbf{t}_b) = \mathbf{t}_a \cdot \mathbf{t}_b + (\sum_{i \in U_a} \Delta \mathbf{t}_{ai} \times \mathbf{t}_{bi}) + (\sum_{i \in U_b} \mathbf{t}_{ai} \times \Delta \mathbf{t}_{bi}) \\ + (\sum_{i \in U_{ab}} \Delta \mathbf{t}_{ai} \times \Delta \mathbf{t}_{bi})\end{aligned} \tag{4}$$

where U_{ab} contains indices of tag a and tag b for which the values are simultaneously updated, and U_a and U_b represent the indices of tag a or tag b for which the values are updated, but not for the other tag (i.e., b and a, respectively).

The Euclidean norm is updated with the following equation for each changed co-occurrence:

$$||\Delta \mathbf{t}_a + \mathbf{t}_a|| = \sqrt{||\mathbf{t}_a||^2 + \sum_{i \in U_a} (\Delta \mathbf{t}_{ai} + \mathbf{t}_{ai})^2 - \sum_{i \in U_a} \mathbf{t}_{ai}^2} \tag{5}$$

In case new tags are added, the dot-product of the new tag with the existing tags is calculated using the normal dot-product formula, shown in Equation 6, since there is no previous information available to calculate this using the difference with respect to previous iterations.

$$\mathbf{t}_a \cdot \mathbf{t}_b = \sum_{i=0}^{n} \mathbf{t}_{ai} \times \mathbf{t}_{bi} \tag{6}$$

The changes of the dot-products between existing tags as result of addition of a new tag can be calculated using the difference (or delta) between the new and old vectors. For this purpose we use Equations 3 or 4 with vectors that have increased with one dimension due to the new tag. Also in this case only the new part is calculated and later added to the existing dot-product of the two tags. Equation 5 should be used to update the Euclidean norm for already existing tags in case a new tag is added, where vectors have increased with one dimension due to the new tag. The original equation for the Euclidean norm, given in Equation 7, should be used to calculate the Euclidean norm for new tags.

$$||\mathbf{t}_a|| = \sqrt{\sum_{i=0}^{n} \mathbf{t}_{ai}^2} \tag{7}$$

After all the changes are processed, the cosines for the changed tag combinations are recalculated using the updated dot-product and the updated Euclidean norm.

4 Implementation

In this section we discuss the implementation of our proposed approaches. First, we describe the data cleaning process, after which we present the incremental recalculation approach and the delta cosine approach for computing cosines. The implementation of the incremental cosine approaches is done in Java and we make use of MySQL for data storage. The data processing is done in PHP.

4.1 Data cleaning

The data we used was made available by Li [8]. It contains 3.5 million unique pictures and 570,000 tags, gathered from the photo sharing site Flickr.com. Since this data set contained some noise, we had to perform data cleaning. The following steps were performed:

- Remove tags longer than 32 characters. This operation is necessary, since we only want words and not complete sentences. Probability of sentences occurring in multiple pictures is quite low and therefore it does not have any effect on our result and therefore can be considered as noise.
- Remove tags that contain non-Latin characters. These tags are removed, since we only focus on English text in this paper.
- Pictures with no tags. Since our approach works with tags, we cannot process pictures without tags.
- Remove tags with low occurrence. We use the formula $\mu - 1.5 \times IQR$, where μ is the average tag occurrence and IQR is the inter-quartile range of the tag occurrences, to determine the minimum number of times a tag has to occur. In the original STCS approach [4], this threshold was set once for the complete data set. Here we calculate the minimum number of occurrences for a tag for every incremental data set and the initial data set. This cleaning rule is only applied to new tags. Already existing tags from previous iterations with a low occurrence in a certain iteration are not affected by this rule.

In order to calculate the cosines for the initial data set and to compute all cosines after updates for reference purposes, we implemented a separate cosine calculator. This program can calculate the cosine similarity for all tag combinations for a given co-occurrence matrix. The implementation was done in Java and for the matrix we used our own customized version of the JAMA [1] matrix library. This customized version uses a one-dimensional array rather than a two-dimensional array to store matrices. Compared to the original implementation of JAMA, our implementation offers faster matrix access with a lower memory footprint.

4.2 Incremental recalculation approach

The incremental recalculation approach implementation consists of three steps. The first step is the import of the new data of the current iteration, where the new data is cleaned using the cleaning rules described in Section 4.1 and then stored in the database. The new pictures are stored in a temporary table. This temporary table allows us to easily create a co-occurrence matrix with only the differences in co-occurrence of tags appearing in existing tags and the co-occurrences of new tags.

The next step in the process is the selection of cosines that need updating. This happens for each non-zero value in a matrix containing the co-occurrences of tags included in the new pictures. A unique list of tag pairs for which the cosine needs to be recalculated is the output of this step.

The last step is the recalculation of cosines from the list of tag pairs. Here we make use of the cosine calculator from the previous section, only instead of calculating every cosine, it uses the list of the previous step to calculate the correct cosines.

4.3 Delta cosine approach

In the delta cosine approach the import of new pictures is the same as in the previous approach. However, the initial import for the delta cosine approach differs from the recalculation approach. The cosine calculator, the dot-product calculator, and a script to store the Euclidean norm to database, have to be run for the initial import of pictures.

The next step is to compute the changes to the cosines. First, we calculate the changes to the dot-product and the changes to the Euclidean norm due to the tag co-occurrence updates of existing tags. Then, the dot-products and Euclidean norms for the new tags (if any) are calculated and the changes to dot-product and Euclidean norm of the existing tags caused by the addition of new tags are determined. After the update and new tag operations are finished, the affected cosines are recalculated using the updated dot-products and the Euclidean norms.

5 Evaluation and results

In this section we evaluate the two proposed incremental cosine computation approaches. First, we discuss the evaluation set-up and the used data set. Then, the performance of each approach is discussed and compared to the baseline approach, which recalculates all cosines each time new pictures are added.

5.1 Evaluation set-up & data structure

The performance of the incremental approaches is measured using the execution time. By considering the execution time, we are able to investigate if there is any performance gain when using our proposed approaches for the incremental computation of cosine similarities. The test is done using a subset of the data set of Li [8]. The initial data set consists of the first 50,000 pictures (1,444 tags) in the original data set.

The incremental data set is composed of randomly selected pictures from the remaining data set. We chose to use 8 incremental data sets of different sizes, which are shown in Table 5. In this table, we give also the number of new pictures as percentage of the total number of pictures, including the corresponding co-occurrence update counts. The reason that we chose for relatively small data set sizes is that we needed to compute all cosine similarities for the baseline, which is a computationally-intensive process. Using these 8 data set sizes, we are able to determine at which point it becomes feasible to use an incremental cosine computation approach. The execution of the complete evaluation is done on a computer with an Intel Core i5 480M with 4 gigabyte of RAM.

5.2 Results of the incremental cosine approaches

The execution time for each approach is measured for each set of newly added pictures. This allows us to compare it to the complete recalculation of all cosines.

Table 5. Properties of the different incremental data sets.

New pictures	2,500	5,000	12,500	25,000	37,00	50,000	62,500	75,000
New pictures (%)	5%	10%	25%	50%	75%	100%	125%	150%
New tags	1	22	183	712	1,408	2,193	2,890	3,682
Total number of tags	1,445	1,466	1,627	2,156	2,852	3,637	4,334	5,126
Updated co-occurrences	10,319	33,730	52,351	79,482	98,643	112,905	137,723	140,145
Updated co-occurrences (%)	1.0%	3.2%	5.0%	7.6%	9.5%	10.8%	13.2%	13.5%

Table 6 shows the execution times for the incremental recalculation approach and complete recalculation approach of the cosine similarity, as well as the speed-up that is obtained by using the incremental calculation approach. This speed-up is used for the comparison with the other approach. The incremental recalculation is faster for all sets. If 2,500 new pictures are added it is 1.44 times faster to use an incremental recalculation than a complete recalculation. For 75,000 new pictures this is 1.23 times faster.

Table 6. Comparison in performance between incremental recalculation and complete recalculation.

Number of new pictures	2,500	5,000	12,500	25,000	37,00	50,000	62,500	75,000
Time incremental calculation (s)	16	17	24	58	144	304	551	922
Time complete recalculation (s)	23	23	31	74	179	378	677	1 137
Speed-up	1.44	1.35	1.29	1.27	1.24	1.24	1.23	1.23

Table 7 shows the execution times for the delta cosine approach and complete recalculation approach of the cosine similarities. The delta cosine approach is faster for all data sets of newly introduced pictures. If 2,500 new pictures are added it is nearly 23 times faster to use incremental recalculation over complete recalculation. For 75,000 new pictures this is 1.24 times faster.

Table 7. Comparison in performance between the delta cosine approach and complete recalculation.

Number of new pictures	2,500	5,000	12,500	25,000	37,00	50,000	62,500	75,000
Time delta cosine (s)	1	1	8	39	120	288	538	919
Time complete recalculation (s)	23	23	31	74	179	378	677	1 137
Speed-up	23	23	3.9	1.9	1.49	1.31	1.26	1.24

Figure 1 shows a plot of the execution times that are shown in Tables 6 and 7. As we can see, the delta cosine approach performs best on all data sets. However, the execution time of the incremental recalculation approach is approaching the execution time of the delta cosine approach for larger updates. This can be explained by the fact the delta cosines approach becomes similar to the incremental recalculation approach when the vectors are changed in many positions. As we

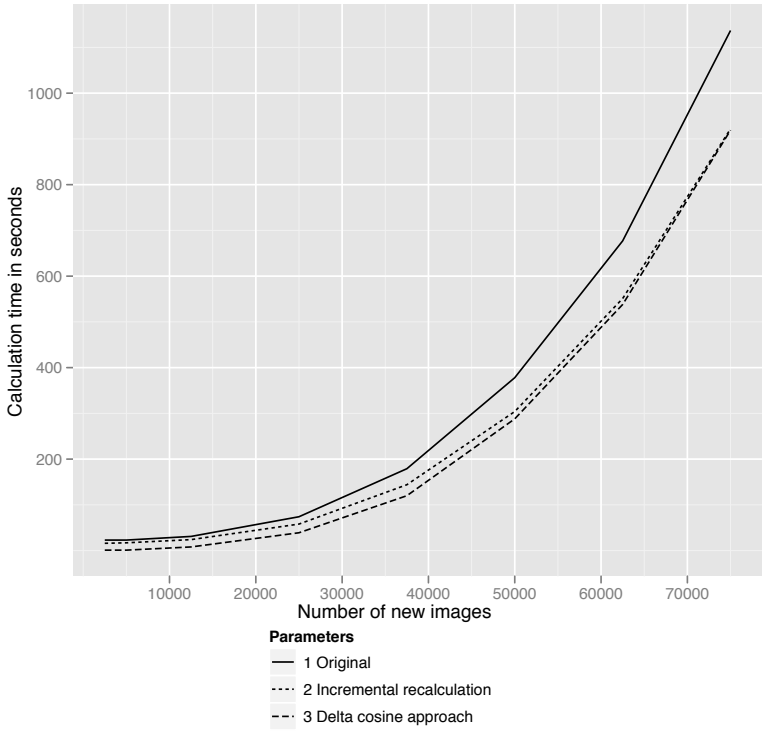

Fig. 1. Execution times for the different approaches.

can observe in Figure 1, the delta cosine approach is more interesting if a small part of the co-occurrences change (for a data set size of 2,500, the delta cosine approach is faster). We can also see in Figure 1 that the delta cosine approach and the incremental recalculation approach are showing an exponential growth in execution time with respect to the number of pictures that are being added. However, this happens also for the complete recalculation approach and we can also notice that our proposed approaches grow with a smaller factor than the complete recalculation approach.

6 Conclusion

In this paper, we propose a method for the incremental calculation of the cosine similarity measure, originating from a scalability problem for tag spaces on the Web. We proposed two approaches for this purpose. The first approach is the incremental recalculation approach. In this approach, we consider updating only the cosine values that are affected by changes in the tag co-occurrences, and cosines that needed to be calculated for the first time, because new tags were

added. The speed-up for the incremental recalculation was between 1.23 and 1.44 when comparing it to a complete recalculation of all cosine similarities.

The second approach is the delta cosine approach and improves on the first approach. In this solution, we calculated the change to the cosine similarity for each change in the co-occurrences matrix and each added tag. Compared to the complete recalculation, the speed-up was 23 for few changes in the co-occurrences and 1.23 for many changes to the tag co-occurrences matrix.

In future work, we would like to investigate how both approaches perform when sequentially adding new pictures. It would also be useful to research how one can perform the syntactic and semantic clustering techniques of the STCS framework in an incremental fashion. Furthermore, we would like to add our proposed approaches to the STCS framework and evaluate these techniques in a setting with a real-time flow of new pictures.

References

1. Java matrix package, http://math.nist.gov/javanumerics/jama/
2. Begelman, G.: Automated Tag Clustering: Improving Search and Exploration in the Tag Space. In: Collaborative Web Tagging Workshop at WWW 2006 (2006), http://www2006.org/workshops/#W06
3. Chakrabarti, D., Kumar, R., Tomkins, A.: Evolutionary Clustering. In: 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD 2006). pp. 554–560. ACM (2006)
4. van Dam, J.W., Vandic, D., Hogenboom, F., Frasincar, F.: Searching and Browsing Tag Spaces Using the Semantic Tag Clustering Search Framework. In: Fourth IEEE International Conference on Semantic Computing (ICSC 2010). pp. 436–439. IEEE Computer Society (2010)
5. Friedman, M., Last, M., Makover, Y., Kandel, A.: Anomaly Detection in Web Documents Using Crisp and Fuzzy-based Cosine Clustering Methodology. Information Sciences 177(2), 467–475 (2007)
6. Gionis, A., Indyk, P., Motwani, R.: Similarity Search in High Dimensions via Hashing. In: 25th International Conference on Very Large Data Bases (VLDB 1999). pp. 518–529. Morgan Kaufmann Publishers Inc. (1999)
7. Jung, S.Y., Kim, T.S.: An Agglomerative Hierarchical Clustering Using Partial Maximum Array and Incremental Similarity Computation Method. In: IEEE International Conference on Data Mining (ICDM 2001). pp. 265–272. IEEE Computer Society (2001)
8. Li, X.: Flickr-3.5M Dataset (2009), http://staff.science.uva.nl/~xirong/index.php?n=DataSet.Flickr3m
9. Radelaar, J., Boor, A.J., Vandic, D., van Dam, J.W., Hogenboom, F., Frasincar, F.: Improving the Exploration of Tag Spaces Using Automated Tag Clustering. In: International Conference on Web Engineering (ICWE 2011). pp. 274–288. Springer (2011)
10. Specia, L., Motta, E.: Integrating Folksonomies with the Semantic Web. In: 4th European Semantic Web Conference (ESWC 2007). Lecture Notes in Computer Science, vol. 4519, pp. 503–517. Springer (2007)
11. Vandic, D., van Dam, J.W., Hogenboom, F., Frasincar, F.: A Semantic Clustering-Based Approach for Searching and Browsing Tag Spaces. In: 26th ACM Symposium on Applied Computing (SAC 2011). pp. 1693–1699. ACM (2011)