Interactive and Context-Aware Tag Spell Check and Correction

Francesco Bonchi
Yahoo! Research
Barcelona, Spain
bonchi@yahoo-inc.com

Ophir Frieder
Georgetown University
Washington DC, USA
ophir@cs.georgetown.edu

Franco Maria Nardini,
Fabrizio Silvestri,
Hossein Vahabi
ISTI-CNR
Pisa, Italy
{first.last}@isti.cnr.it

ABSTRACT

Collaborative content creation and annotation creates vast repositories of all sorts of media, and user-defined tags play a central role as they are a simple yet powerful tool for organizing, searching and exploring the available resources. We observe that when a user annotates a resource with a set of tags, those tags are introduced one at a time. Therefore, when the fourth tag is introduced, a knowledge represented by the previous three tags, i.e., the context in which the fourth tag is produced, is available and exploitable for generating potential correction of the current tag. This context, together with the "wisdom of the crowd" represented by the co-occurrences of tags in all the resources of the repository, can be exploited to provide interactive tag spell check and correction. We develop this idea in a framework, based on a weighted tag co-occurrence graph and on nodes relatedness measures defined on weighted neighborhoods. We test our proposal on a dataset coming from YouTube. The results show that our framework is effective as it outperforms two important baselines. We also show that it is efficient, thus enabling its use in modern tagging services.

Categories and Subject Descriptors: H.3.1 [Information Storage and Retrieval]: Content Analysis and Indexing – *Linguistic processing*

Keywords: tag spell checking and correction; tag co-occurrence graph.

1. INTRODUCTION

Collaborative tagging services are one of the most distinguishing features of Web 2.0. Flickr, YouTube, del.icio.us, Technorati, Last.fm, or CiteULike – just to mention a few – as they allow their users to upload a photo, to publish a video, to share an interesting URL or to bookmark a scientific paper, and provide them the possibility to assign tags, i.e., freely chosen keywords, to these resources. Such a collaborative content creation and annotation effort creates vast repositories of all sort of media. User-defined tags play a central role there as they are a simple yet powerful tool for organizing, searching and exploring the resources. Obviously, all these applications need to assume that tags are "correct". However, this assumption is not realistic in the real world as tags are noisy, contain typos, and many different spellings can be used for the same concept (e.g., the term "*hip hop*" and its variants "*hip-hop*", "*hiphop*", or "*hip_hop*"). It is thus important to develop systems to help users to provide correct and well-established tags, so as to improve the overall quality of annotations. For instance, correcting *"hip hop"* as *"hip-hop"*, when the latter is more frequent than the former, is useful because this keeps the labeling of the concept uniform, allowing for an improved findability and a better organization/exploration of the associated resources. More important, by correcting for example, *brittny* with *britney*, we allow the given resource to be found by means of the correct spelling of the word. This, in turn, is not true if the correction is not provided and, therefore, the resource can not be retrieved.

A similar problem is dealt with in the context of Web search engines by exploiting query spell checkers. However, while query correction exploits the position of words within the query, in the case of tag systems, words position is not meaningful, as an annotation is a set of tags and not a sequence. What is meaningful instead is the context in which a tag is used, that is, the other tags in the annotation of a resource. If we know that people tagging an object with *"apple"* are more likely to tag *"store"* instead of *"str"*, then we could suggest the former as a possible spell correction for the latter.

Our working hypothesis is to spell check and correct within the tagging process once a tag is completely written. The process thus becomes to analyze each tag while the tagging process of the user is still going on.

We model the "wisdom" of all the users of a collaborative tagging system by means of a weighted co-occurrence graph to develop a context-aware tag spell checker and corrector able to interactively check and emend terms to the user.

As an example, we extract a portion of the tag co-occurrence graph of YouTube for the tag *"brittny"* in two different contexts. The first context is {*Circus, Pop, Video*}, while the second one is {*HappyFeet, Music*}. The example shows how the same tag can be corrected in two different ways depending on the context. In the first case, the user is clearly referring to Britney Spears, as one of the tag ("Circus") is the title of a Britney Spears song. While in the second case the user

CIKM'12, October 29–November 2, 2012, Maui, HI, USA.

is referring to the actress and singer Brittany Murphy, who gave voice to one of the penguins in the computer-animated movie *"Happy Feet"*, singing also two songs for the movie. Note that, beyond correcting the misspelled tags, context-awareness in a tag co-occurrence graph, might also be used to suggest other meaningful tags, e.g., *"Spears"* in the first context, and *"Murphy"* in the second.

2. RELATED WORK

We focus on two main research issues: *spell checking* and the use of *contextual* information to support an interactive tag spelling correction. Here, we briefly summarize the key results in both the two fields.

Research on spell checking has focused either on non-word errors or on real-word errors [5]. Non-word errors such as *ohuse* for *house* can easily be detected by validating each word against a lexicon, while real-world errors, e.g., *out* in *I am going our tonight*, are difficult to detect. Current context-sensitive spelling correctors that are required for real-world errors mainly rely on two kinds of features: *collocation*, and *co-occurrence* [4, 7]. Both approaches generate a high-dimensional feature space. They have been used only for short lists of commonly confused words.

Cucerzan *et al.* [2] investigate the use of implicit and explicit information about language contained in query logs for spelling correction of search queries. They present an approach that uses an iterative transformation of the input query string into sequence of more and more likely queries according to statistics extracted from query logs. Chung *et al.* [10] propose an approach to spelling correction based on Aspell. The method re-ranks the output of Aspell and then a discriminative model (Ranking SVM) is employed to improve upon the initial ranking. Merhav *et al.* [8] use a probabilistic approach to enrich descriptors with corrected terms in a P2P application. To use as much information as possible, they include features from character level, phonetic level, word level, syntax level, and semantic level. These are evaluated by a support vector machine to predict the correct candidate. Authors test the correction capabilities of their system by comparing it with others spelling correctors (i.e., Microsoft Word, Google, Hunspell, Aspell, FST) showing that their system outperforms in recall by at least 3% even if confined to non-word errors.

Recently, Gao *et al.* [3] focus on exploiting noisy Web corpora and query logs for spelling correction purposes. In particular, authors propose a distributed infrastructure for training and applying Web scale n-gram language models, and a phrase-based error model, where the probability of transformations between multi-word phrases is estimated using query-correction pairs derived from search logs. Bao *et al.* [1] present an algorithm that is based on statistics from the corpus data (rather than the query log). More in detail, authors propose a graph based spelling correction approach, that can incorporate different types of data sources with different levels of reliability.

Nardini *et al.* [9] build a tag spell checker using a graph-based model. In particular, the authors present a technique based on the graph of tags associated with objects made available by online sites such as Flickr and YouTube. Authors show the effectiveness of the approach on the basis of experimentation done on real-word data. The goal pursued in [9] is to use the neighbors of the wrong tag to find the correct version of the current tag. This is done by using a tag co-occurrences graph and by exploiting a combination of edit distance and some simple graph properties. We approach the problem from a different point of view. We build a co-occurrence graph of only correct tags (easy to build by using public data and by using techniques as presented in [9]), and we match an incoming tag to the nodes in the graph to understand if it is misspelled and, if so, what is its correction to suggest to the user by exploiting link-prediction measures. An important difference between our method and the one proposed in [9] is that, while the model in [9] uses only the current tag to devise possible correct versions of it, our method is also able to exploit the "context", i.e., other tags that have been already assigned by the user, to carry out the correction process. Furthermore, our method incrementally updates the spell check and correction model.

3. MODEL

In this section we introduce our model, which is a weighted co-occurrence graph model to capture relationships among tags, and we describe how we exploit such relationship for context-aware tag spell check and correction.

Weighted Tag Co-occurrence Graph: Let R be a set of resources. Let Σ be a finite alphabet. Let $T \subseteq \Sigma^*$ be a set of tags associated with each resource and let $\gamma : R \to T$ be a function from resources to set of tags mapping a resource with its associated set of tags. Furthermore, we define $T^* = \cup \{\gamma(r), \forall r \in R\}$ to be the union of all tags for all resources in R.

Let $G = (V, E)$ be an undirected graph. V is the set of nodes where each node represents a tag $t \in T^*$, and E is the set of edges defined as $E = V \times V$. Given two nodes, u, v, they share an edge if they are associated at least once with the same resource. More formally, $E = \{(u, v) | u, v \in V,$ and $\exists r \in R | u, v \in \gamma(r)\}$. We denote $N(v)$ the immediate (distance 1) neighborhood of a node v, i.e., $N(v) = \{t \in V | (v, t) \in E\}$.

Both edges and nodes in the graph are weighted. Let $u, v \in V$ be two tags. Let $w_e : V \times V \to \mathbb{R}$ be a weighting function for edges measuring the co-occurrence of the two tags, namely, the number of times the two tags appear together for a resource. For convenience, we assume $w_e(u, v) = 0$ when $(u, v) \notin E$. For a given node $v \in V$, $w_v : V \to \mathbb{R}$ associates a tag with its weight. We will see later that weights of nodes and edges can be used to rank the results and to reduce the size of the graph G discarding irrelevant edges and nodes, thus gaining in efficiency.

We assume that the graph does not contain misspelled tags. This can be achieved by correcting the graph off-line as in [9], and on-line, by incrementally updating the graph with correctly spelled tags.

Tag and Tag-context: In collaborative tagging services, users annotate a resource with a set of tags, not a sequence. An observation behind our model is that, although we are dealing with a set of annotations and not a sequence, the user provides this set of annotations by introducing tags one by one. Following this observation in our model, we consider as input a sequence of tags $\langle t_1, \ldots, t_{n-1}, t_n \rangle$, in the order as they are introduced by the user. In this setting t_n is the last tag introduced, which is going to be spell-checked by our model, while the set $\{t_1, \ldots, t_{n-1}\}$ refers to its context, that we denote also as $C(t_n)$. We assume that the context is correct, as we correct tags one by one

while they are introduced. Moreover, we assume that at the moment we process t_n all tags in the context already belong to the graph, i.e., $C(t_n) \subseteq V$. This is indeed guaranteed by our model as a correct tag $t \notin V$ is added to the graph as shown later.

Tag Spell Check and Correction: We are given the weighted tag co-occurrence graph G, the last introduced tag t_n and its context $C(t_n) = \{t_1, \ldots, t_{n-1}\}$.

All tags in the context $C(t_n)$ are present in the graph. Therefore, we can define the set of candidate tags to be considered as a neighborhood of the nodes in $C(t_n)$. More formally, we define $\mathcal{N}(C(t_n)) = \bigcup_{t \in C(t_n)} N(t)$ and we consider the subgraph of G induced by $\mathcal{N}(C(t_n))$. In this induced subgraph we select the best node to be suggested as correction. This selection is done on the basis of relatedness between a candidate tag $u \in \mathcal{N}(C(t_n))$ and the original tag t_n that we want to correct. It is important to note that the tag t_n is temporarily assumed to be linked to its context, thus it belongs to $\mathcal{N}(C(t_n))$ and, consequently, to the induced subgraphs.

For defining relatedness we adopt graph neighborhood-based measures that have been successfully used, e.g., for link prediction [6]. We report the ones we use in Table 1.

For sake of efficiency, the set of tags that we consider as candidate spell corrections are a subset of $\mathcal{N}(C(t_n))$, governed by two system parameters r and δ. In particular, we consider the subgraph induced by $\mathcal{N}(C(t_n))$. We sparsify it by keeping only the top-r edges with respect to their weights w_e. We then further sparsify the candidate nodes by keeping only nodes with edit distance (Damerau-Levenshtein distance[1]) no more than a given threshold δ.

Now, what if the given tag t_n is correct? If the tag is correct, it is likely to have appeared before, thus it is already part of the graph. Moreover, it is also quite likely that it appeared at least once with one other tag of the current context. Our model is able to detect these cases and decide when it is not needed to spell check. The spell check mechanism comes into play if the tag is not part of the graph-based model and it has never been seen before within that context. If the user accepts the proposed correction, the model is updated with the new information regarding the last seen co-occurrences while, if the user does not accept the proposed correction, the tag is considered "correct" and the method adds it to the graph updating the whole model.

4. EXPERIMENTS

We assess the effectiveness and the efficiency of our spell check and correction technique. We run our experiments on a dataset of tags obtained by crawling 568,458 distinct videos from YouTube. We obtain a total of 5,817,896 tags. We preprocess the dataset by removing noisy tags (i.e., the ones having a low frequency). After this preprocessing step, the dataset contains $4,357,971$ tags, $434,156$ of which are unique[2].

We build our tag spell check and correction model by using the dataset described above, from which we also sample a test set of 250 videos that we denote $O = \{o_1, o_2, \ldots, o_{250}\}$. For each video $o \in O$, let $\gamma(o)$ be the set of tags associated with the video o. We evaluate the performance of our proposed method on this test set of videos. In particular, we preprocess the test set by checking, for each video o, the presence of some misspelled tags in the set $\gamma(o)$. We use two methods: we first apply the technique proposed in [9], and we also ask three assessors to assure by manually checking that all tags are correctly-spelled. The test set is then split in ten sets $O_1, \ldots, O_{10}$ each containing 25 of the videos in O. For each video in $O_1, \ldots, O_5$, we randomly choose one tag and we replace it with a wrong one having a Damerau-Levenshtein distance lower or equal to a value δ with respect to the original one. We use values of δ in the set $\{1, 2\}$. Furthermore, for each video in $O_6, \ldots, O_{10}$, we randomly choose one of its tags, and we replace it with a new one listed in the page of common misspelled tags[3].

By exploiting the experimental framework described above, we are going to answer the following research questions: 1) How the proposed approach behaves in terms of effectiveness?, 2) What is the link-prediction measure demonstrating the best performances in solving the tag spell check and correction problem?, 3) Is the proposed approach efficient? Is it possible to exploit it in modern real-world tagging services?

Effectiveness: Here, we are going to answer the first two research questions. We do this by testing the precision obtained by our approach using different link-prediction measures in the ranking of the candidate corrections. Let x be the number of videos in $O_1, \ldots, O_5$ and $O_6, \ldots, O_{10}$ that receives a "correct" tag spelling correction, and y be the total number of videos that receive at least one tag spelling correction. We compute the *precision* (x/y) for each of the ten sets: $O_1, \ldots, O_5$ and $O_6, \ldots, O_{10}$. As we are evaluating the capabilities of the system in detecting and correcting wrong tags, in this evaluation we consider only one (i.e., the top-1 ranked) correction proposed. Furthermore, we define a *coverage* measure for $O_1, \ldots, O_5$ and $O_6, \ldots, O_{10}$, as the number of videos in the test set that receive a "correct" tag spelling correction divided by the total number of video composing each set (25) to obtain the percentage of the set covered by "correct" tag spelling corrections. In Table 2, we report for each proposed method and dataset used, both the average precision (%) and the average coverage (%) together with their variance.

We compare our results against two baselines: the first one (i.e., "Damerau-Levenshtein") is obtained by computing the Damerau-Levenshtein distance between the wrong tag and all the nodes at distance one from its context, while the second one is obtained by using the technique proposed in [9] (we refer to it with the name "graph pruning").

The evaluation is performed by asking the different systems for possible corrections and we consider it as "correct" if the output of the system consists of only one tag and it is correct. We test our method with the proposed ranking schemes on both the two group of sets $O_1, \ldots, O_5$ and $O_6, \ldots, O_{10}$.

Our technique shows good performance when δ is equal to 1 both in $O_1, \ldots, O_5$ and $O_6, \ldots, O_{10}$. In this case, the method exploiting the *common neighbors* metric shows the best performance in terms of average precision and average coverage on the two groups of sets considered ($O_1, \ldots, O_5$

[1] `http://en.wikipedia.org/wiki/Damerau-Levenshtein_distance`

[2] The two versions of the datasets can be downloaded here: `http://hpc.isti.cnr.it/~vahabi/tag.tgz`

[3] `http://en.wikipedia.org/wiki/Wikipedia:Lists_of_common_misspellings`

and $O_6, \ldots, O_{10}$). Furthermore, while *common neighbors* scores the best, three methods (i.e., *weighted common neighbors*, *preferential attachment*, and *weighted preferential attachment*) perform always as second, third and fourth (with a small exception for *weighted common neighbors* where its performance in terms of average coverage is equal to *common neighbors*). We explain this behavior by observing that *common neighbors* and *weighted common neighbors* work by exploiting how the current node and its possible corrections are linked in the graph. Therefore, *preferential attachment*-based metrics only take care of the importance of the current node. *Common neighbors*-based metrics are thus able to exploit the knowledge represented in the graph in a more effective way.

A different behavior is exhibited for δ equal to 2. Here, the *weighted common neighbors* metric shows the best performances in terms of average precision and average coverage on the two group of sets considered ($O_1, \ldots, O_5$ and $O_6, \ldots, O_{10}$). Furthermore, the three methods (i.e., *common neighbors*, *weighted preferential attachment*, and *preferential attachment*) perform always as second, third and fourth. Differently from the case where δ is equal to 1, the weighted versions of the *common neighbors* and *preferential attachment* metrics work better than their unweighted versions. The rationale of this behavior could be explained by observing that, when δ is equal to 2, the number of selected nodes from the graph as candidate spelling corrections is greater than in the previous case ($\delta = 1$). The weighted versions of the two metrics are thus more robust to possible noise deriving from extending the set of spelling correction candidates.

Jaccard-based metrics provide low scores in all the tests conducted. In particular, the performances on the group of sets $O_6, \ldots, O_{10}$ for both the two values of δ are lower than the two baselines. This behavior could be explained by observing that *Jaccard*-based metrics take into account the union of the neighborhood of the current node and its possible correction. Consequently, if the graph contains noise (in the form of nodes with low-weight edges), this metric better supports that phenomenon.

We also highlight that the variance associated to our metrics for all the tests conducted is low. In particular, it is always lower than 1.2%. It reveals a high stability of results with respect to variations of the dataset.

To conclude, the best performances on both the two group of sets ($O_1, \ldots, O_5$ and $O_6, \ldots, O_{10}$) in terms of average precision and average coverage are obtained by using *common neighbors*-based metrics with δ equal to 1.

Efficiency: We evaluate the response time of both the baseline and our spell check and correction method. We measure the time needed by the system to produce the list of corrections for a given tag using the *common neighbors* metric, which is the best scoring method when $\delta = 1$ (see Table 2). Tests are conducted using a PC with a 1.66GHz CPU and 4GB of RAM. All the algorithms are implemented in Java.

We analyze the response time of the system by varying the value of a filtering threshold r that is used to prune edges of the tag co-occurrence graph, G. For each node in G, we preprocess the graph by keeping only the top-r scoring edges and by removing the remaining ones. The pruning of the tag co-occurrence graph speeds-up the spell check and correction process degrading, though, its effectiveness. Table 3 shows the average precision (%) and the average coverage (%) – computed over all the sets, $(O_1, \ldots, O_{10})$ – for both our method and the baseline. When no pruning is performed on the graph ($r = \infty$), our method corrects a misspelled tag in 33 milliseconds while the same operation is much faster when the graph G is pruned. In particular, by setting r equal to 200, our method corrects a wrong tag in about 2 milliseconds being 16 times faster than the basic case (original graph) with only 10% precision and coverage loss. The low response time of the entire tag spell check and correction process allows the use of our proposed technique within modern tagging services.

5. CONCLUSIONS AND FUTURE WORK

We introduced a model for context-aware, interactive, tag spell check and correction. Our described approach exploits this context together with the available co-occurrences of tags in all the resources of the repository to provide an interactive tag spell correction. Experiments on a dataset of tags coming from YouTube videos show that our method is effective and outperforms two important baselines. Furthermore, we also prove that the method is efficient as it is able to check and correct a tag in two milliseconds with a good trade-off in terms of average precision and average coverage.

In our future investigation, we intend to increase the degree of interactivity of the system by enabling "as-you-type" tag spell check and correction.

6. ACKNOWLEDGMENTS

This research has been partially funded by the EU CIP-ICT PSP 2011.4.1 InGeoCloudS Project (Grant Agreement no. 297300), and by the EU FP7-ICT-2009-5 CONTRAIL Project, (Grant Agreement no. 257438).

7. REFERENCES

[1] Z. Bao, B. Kimelfeld, and Y. Li. A graph approach to spelling correction in domain-centric search. In *Proc. HLT'11*, pages 905–914, Stroudsburg, PA, USA, 2011. ACL.

[2] S. Cucerzan and E. Brill. Spelling correction as an iterative process that exploits the collective knowledge of web users. In *Proc. EMNLP*, 2004.

[3] J. Gao, X. Li, D. Micol, C. Quirk, and X. Sun. A large scale ranker-based system for search query spelling correction. In *Proc. COLING'10*, pages 358–366, Stroudsburg, PA, USA, 2010. ACL.

[4] A. R. Golding and D. Roth. A winnow-based approach to context-sensitive spelling correction. *Mach. Learn.*, 34(1-3):107–130, 1999.

[5] K. Kukich. Techniques for automatically correcting words in text. *ACM Comput. Surv.*, 24(4):377–439, 1992.

[6] D. Liben-Nowell and J. Kleinberg. The link-prediction problem for social networks. *JASIST*, 58(7):1019–1031, 2007.

[7] L. Mangu and E. Brill. Automatic rule acquisition for spelling correction. In *Proc. ICML'97*, pages 187–194, San Francisco, CA, USA, 1997. Morgan Kaufmann Publishers Inc.

[8] Y. Merhav and O. Frieder. On multiword entity ranking in peer-to-peer search. In *Proc. SIGIR'08*. ACM, 2008.

[9] F. M. Nardini, F. Silvestri, H. Vahabi, P. Vahabi, and O. Frieder. On tag spell checking. In *Proc. SPIRE'10*, SPIRE'10, pages 37–42. Springer-Verlag, 2010.

[10] C. Whitelaw, B. Hutchinson, G. Y. Chung, and G. Ellis. Using the web for language independent spellchecking and autocorrection. In *Proc. EMNLP'09*. ACL, 2009.

Metric	Unweighted Version	Weighted Version
preferential attachment	$\|N(u)\|$	$\sum_{z \in N(u)} w_e(u,z)$
common neighbors	$\|N(u) \cap N(t_n)\|$	$\sum_{z \in N(u) \cap N(t_n)} w_e(u,z) + w_e(t_n,z)$
jaccard of neighbors	$\frac{\|N(u) \cap N(t_n)\|}{\|N(u) \cup N(t_n)\|}$	$\frac{\sum_{z \in N(u) \cap N(t_n)} w_e(u,z) + w_e(t_n,z)}{\sum_{z \in N(u) \cup N(t_n)} w_e(u,z) + w_e(t_n,z)}$

Table 1: Metrics used in our model for ranking a candidate node u, given that the tag we want to correct is t_n.

Test Set	Ranking Method	avg. Precision (Variance)	avg. Coverage (Variance)	δ
$O_1, \ldots, O_5$	Damerau-Levenshtein	64.77 (1.59)	59.2 (0.91)	1
	graph pruning	70.80 (1.43)	48.6 (0.77)	1
	preferential attachment	81.77 (0.45)	75.2 (0.51)	1
	weighted preferential attachment	80.90 (0.38)	74.4 (0.45)	1
	common neighbors	**85.22** (0.76)	**78.4** (0.85)	1
	weighted common neighbors	85.14 (0.70)	**78.4** (0.93)	1
	jaccard neighbors	68.83 (0.49)	63.2 (0.35)	1
	weighted jaccard neighbors	73.14 (0.78)	67.2 (0.67)	1
$O_1, \ldots, O_5$	Damerau-Levenshtein	35.04 (0.38)	33.6 (0.37)	2
	graph pruning	50.32 (0.38)	48.2 (0.69)	2
	preferential attachment	55.67 (0.80)	51.2 (0.75)	2
	weighted preferential attachment	55.74 (0.90)	51.2 (0.75)	2
	common neighbors	62.75 (0.26)	57.6 (0.13)	2
	weighted common neighbors	**66.12** (0.59)	**60.8** (0.51)	2
	jaccard neighbors	37.37 (0.18)	34.4 (0.21)	2
	weighted jaccard neighbors	46.38 (0.31)	42.4 (0.21)	2
$O_6, \ldots, O_{10}$	Damerau-Levenshtein	67.07 (0.75)	58.4 (0.69)	1
	graph pruning	73.4 (0.32)	66.2 (0.48)	1
	preferential attachment	83.44 (0.47)	72.8 (0.75)	1
	weighted preferential attachment	83.44 (0.47)	72.8 (0.75)	1
	common neighbors	**87.22** (0.5)	**76.0** (0.64)	1
	weighted common neighbors	85.22 (0.45)	74.4 (0.85)	1
	jaccard neighbors	63.34 (0.39)	55.2 (0.43)	1
	weighted jaccard neighbors	76.99 (0.32)	67.2 (0.59)	1
$O_6, \ldots, O_{10}$	Damerau-Levenshtein	27.94 (0.47)	28 (0.48)	2
	graph pruning	32.04 (0.77)	31.05 (0.68)	2
	preferential attachment	47.47 (0.61)	46.4 (0.69)	2
	weighted preferential attachment	51.57 (0.86)	50.4 (0.93)	2
	common neighbors	59.77 (0.9)	58.4 (1.01)	2
	weighted common neighbors	**65.47** (1.40)	**64.0** (1.52)	2
	jaccard neighbors	19.63 (1.14)	19.2 (1.07)	2
	weighted jaccard neighbors	27.87 (0.98)	27.2 (0.91)	2

Table 2: Effectiveness of the proposed methods in terms of average precision (%) and average coverage (%) for the videos in $O_1, \ldots, O_5$ and $O_6, \ldots, O_{10}$.

r	Response time	*common neighbors*		Damerau-Levenshtein	
		avg. Precision (Var.)	avg. Coverage (Var.)	avg. Precision (Var.)	avg. Coverage (Var.)
10	0.30×10^{-3}	29.23 (1.02)	25.61 (1.19)	29.29 (1.69)	24.80 (0.96)
50	0.64×10^{-3}	42.33 (1.15)	36.84 (0.72)	32.05 (1.82)	28.00 (1.56)
100	0.76×10^{-3}	51.30 (0.91)	44.05 (0.83)	42.33 (2.11)	36.80 (1.34)
200	2.06×10^{-3}	77.72 (1.10)	67.27 (0.91)	53.98 (1.78)	44.10 (1.87)
∞	33.22×10^{-3}	85.22 (0.76)	78.40 (0.85)	64.77 (1.59)	59.20 (0.91)

Table 3: Response time (sec.) and effectiveness in terms of average precision (%) and average coverage (%) of our method (using the *common neighbors* metric) and the Damerau-Levenshtein baseline by varying the filtering threshold r over all the sets, $(O_1, \ldots, O_{10})$. A value of r equal to ∞ means that no pruning on the tag co-occurrence graph is performed before the spell check and correction process.