STRec: An Improved Graph-based Tag Recommender

Modou Gueye
Université Cheikh Anta Diop
Dakar, Sénégal
gmodou@ucad.sn

Talel Abdessalem
Institut Telecom - Telecom ParisTech
Paris, France
Talel.Abdessalem@enst.fr

Hubert Naacke
LIP6, UPMC Sorbonne Universités - Paris 6
Paris, France
Hubert.Naacke@lip6.fr

ABSTRACT

Tag recommendation is a major aspect of collaborative tagging systems. It aims to recommend tags to a user for a given item. In this paper we propose an adaptation of the search algorithms proposed in [14, 1] to the tag recommendation problem. Our algorithm, called STRec, provides network-aware recommendations based on proximity measures computed on-the-fly in the network. STRec uses a bounded search to find good neighbors.

On top of STRec, we apply a re-ranking scheme that improves the quality of the recommendations. We update the ranking according to the degree of association between the higher ranked tags and the lower ranked ones. This technique leads to better recommendations as we show in this paper and could be applicable on top of many recommender systems. The experiments we did on several datasets demonstrated the efficiency of our approach.

Categories and Subject Descriptors

H.3.3 [**Information Search and Retrieval**]: Information Filtering

General Terms

Algorithms, Experimentation

Keywords

Graph-Based Tag Recommendations, Social Network, Collaborative Filtering, Association Rules Mining

1. INTRODUCTION

Social (i.e. collaborative) tagging is the practice of allowing users to annotate content. The users can organize, and search content with annotations called tags. The growth of popularity of social media sites has made the area of recommender systems for social tagging systems an active and growing topic of research [12, 17, 9].

Tag recommendation aims to infer the most suited tags to a user for tagging a given item. It is a salient part of the Web 2.0 where applications are user-centered. We present, in this paper, an efficient tag recommender algorithm named STRec. Our algorithm adapts the search algorithms proposed in [14, 1, 24] to the tag recommendation problem, and extends them by a re-ordering step that improves the quality of the recommendations. The basic idea of STRec is to merge two recommendation components: a social network based component, together with a network-independent one. The first component relies on social networks to provide recommendations to a user. It analyses the tags existing in the user's neighborhood; we say that it computes the social frequencies of tags. Then, it retrieves the most frequent tags within the user's neighborhood. The contribution of each neighbors's tag to the final recommendation, is weighted by the proximity of that neighbor (i.e. tagging similarity in our case) with the user. The second component takes into account the global (i.e. network-independent) frequencies of tags. The global frequency represents the popularity of a tag for tagging a given item, whatever the user is. Then, we aggregate these two frequencies in our STRec model, and compute a sorted list of tags to recommend. Finally, relatively to the first tag of the list, the rest of the recommended tags is reordered using association rules, in order to bring more accuracy to the final recommendation. Our experimental results, in Section 4, show that the first tag of the list has indeed a significant benefit on the quality of the recommendation. They also confirm the efficiency of STRec.

The remainder of this paper is organized as follows. In Section 2 we present some preliminaries. Section 3 details the STRec algorithm. In Section 4, we present experimentations of our proposal. Section 5 summarizes the related work, and Section 6 concludes the paper.

2. PRELIMINARIES

A folksonomy is a system of classification that allows users to annotate and categorize content by the way of creating and managing tags. It is related to the event of social tagging systems[1] and can be defined as a collection of: a set of users U, a set of tags T, a set of items I, and a ternary relation between them $S \subseteq U \times I \times T$.

A tagging triple $(u, i, t) \in S$ means that user u has tagged an item i with the tag t. A user can tag an item with one

[1] http://en.wikipedia.org/wiki/Social_bookmarking

or more distinct tags from T. We denote by $T(u,i)$ the set of all distinct tags used by a user u to tag an item i

$$T(u,i) = \{t \in T | \exists (u,i,t) \in S\}$$

On top of this folksonomy, we consider an undirected weighted graph of users $G = (U, E, \sigma)$ called the social network. In G, the nodes represent the users and σ is a function that associates to each edge $e = (u,v) \in E$ a value, $\sigma(u,v) \in [0,1]$, called the proximity (or social) score between u and v.

We assume that a user can tag an item with a given tag at most once. The interest of a tag t for a given user u and an item i can be estimated by a score function $score(t|u,i)$. The score function depends on the recommender's model. Then the "Top-K" highest scoring tags are recommended by

$$Top(u,i,K) = \underset{t \in T}{^{K}argmax}\ score(t|u,i) \quad (1)$$

3. STREC: A GRAPH-BASED TAG RECOMMENDER

We first present the score function and the extended proximity we use in our algorithm, as they were defined in [14], then we present the STRec algorithm.

3.1 Definition of the score function

They model for a user, an item, and a tag triple (u,i,t), the score $score(t|u,i)$ of tag t for the given user u and item i by

$$score(t|u,i) = h(fr(t|u,i)) \quad (2)$$

where $fr(t|u,i)$ is the overall tag's frequency of tag t for user u and item i, and h is a positive monotone function. The overall tag's frequency function $fr(t|u,i)$ is defined as a combination of a network-dependent component and an item-dependent one, as follows:

$$fr(t|u,i) = \alpha \times tf(t,i) + (1-\alpha) \times sf(t|u,i) \quad (3)$$

The first component, $tf(t,i)$, is the tag's frequency of t for i, i.e., the number of times i was tagged with t. The sf component stands for social frequency, an important measure that depends on the neighborhood of user u. In Equation 3, the parameter α allows to tune the relative importance of the social component with respect to tag's frequency. When α is valued 1, the score becomes network-independent. Reversely, when α is valued 0 the score depends exclusively on the social network. Let us notice that we use, as a social network, a similarity network (weighted graph) inferred from the tagging behaviour of the users. The weight associated to each edge is a value between 0 and 1 representing the degree of similarity between two users.

Considering that each user brings her own weight (proximity) to the score of a tag, the measure of tag's social frequency is defined as follows:

$$sf(t|u,i) = \sum_{v \in \{U|(v,i,t) \in S\}} \sigma(u,v) \quad (4)$$

In this formula, v stands for a neighbor of user u who tagged item i with tag t, and $\sigma(u,v)$ is the proximity (edge weight) between u and v.

Extended proximity. The above scoring model takes into account only the neighborhood of a given user (the users directly connected to her). But, this can be extended to deal also with users that are indirectly connected to this user, following a natural interpretation that user links (e.g., similarity) are, at least to some extent, transitive. An extended proximity σ^+ can be deduced from σ for any pair of users connected by a path in the network. Then, σ^+ can replace σ in the definition of social frequency considered before (Equation 4), yielding an overall tag scoring scheme that depends on the entire network instead of only the directly connected neighbors. In the rest of the paper, we consider this extended proximity and denote by user's proximity vector, the list of all her neighbors v (of course we consider $\sigma(u,v) > 0$) ordered in descending order of their proximity values.

The STRec algorithm, as the TOPKS algorithm proposed in [14], computes on-the-fly the proximity values with respect to a given user u. The issue is to facilitate the retrieval of the most relevant unseen user v in the network (i.e. v is not a direct neighbor of u), along with her proximity value $\sigma^+(u,v)$. The user v will have the potential to contribute the most to the partial scores of tags that are still candidates for the most relevant result.

Inspired by studies in the area of trust propagation for belief statements, the weights on a given path $p = (u_1, ..., u_n)$ between u_1 and u_n is aggregated by multiplying them [14].

$$\sigma^+(p) = \prod_i \sigma(u_i, u_{i+1}) \quad (5)$$

The multiplication function is monotonically decreasing over any path it is applied to, when σ draws values from the interval $[0,1]$. Thus given a social network G and a path $p = (u_1, ..., u_n) \in G$, we have $\sigma^+(u_1, ..., u_{n-1}) \geq \sigma^+(u_1, ..., u_n)$.

We can define σ^+ for any pair of connected users (u,v) in the network by taking the maximal weight over all their connecting paths. More formally, $\sigma^+(u,v)$ is defined as

$$\sigma^+(u,v) = \max_{p \in G} \left\{ \sigma^+(p) | u \xrightarrow{p} v \right\} \quad (6)$$

A greedy approach is applicable to allow browsing the network of users on the fly, at recommendation time, visiting them in the order of their proximity with respect to a given user (for whom we want to make recommendations). More precisely, by generalizing Dijkstra's algorithm [5], a max-priority queue (denoted H) is maintained, whose top element $top(H)$ will be at any moment the most relevant unvisited user. A user is visited when her tags are taken into account for the top-k result, which can occur at most once.

At each step advancing in the network, the top of the queue is extracted (visited) and its unvisited neighbours (adjacent nodes) are added to the queue (if not already present) and are relaxed. Relaxation updates the best proximity score of these nodes, as described in Algorithm 1. It can be shown by straightforward induction that this greedy approach allows to visit the nodes of the network in decreasing order of their proximity with respect to a given user. For more details on the scoring model described above, we refer the reader to [14, 1]. The following sections describe how this

Algorithm 1: Relaxation

```
1 if (σ+(u,x) × σ(x,v)) > σ+(u,v) then
2 |  σ+(u,v) ← (σ+(u,x) × σ(x,v))
3 end
```

greedy procedure for iterating over the network is used in our recommendation algorithm.

3.2 The STRec algorithm

As already introduced in previous sections, the network-dependent component is an important part of the STRec algorithm.We mostly focus on the computation of the social frequency, $sf(t|u,i)$, as it is a key parameter in the scoring function of tags.

First, a list D of top-k candidate tags is kept and sorted in descending order of their minimal possible scores (we define shortly). A tag becomes candidate when it is met for the first time in a tagged triple.

The algorithm 2 presents the computation of the STRec network-dependent component for a given user u for whom we want to recommend tags for an item i. For each of her (direct or indirect) neighbors v on the social network we retrieve the list of tags this neighbor employed for the item i. Then for each of these tags t , we update its social frequencie $sf(t|u,i)$ and we add it in D, as candidate tags if it is not already in the top-k candidates list (lines 1 to 8).

We assume that, for item i, we have an inverted list $IL(i)$ of the tags t used to tag i, along with the corresponding tag frequencies $tf(t,i)$ in a descending order of these frequencies. Starting from the top most frequent tag, this list will be consumed one tag at a time, whenever the current tag becomes candidate for the top-k result (lines 9 to 17).
By $CIL(i)$ we denote the tags already consumed in $IL(i)$ (as known candidates), by $top\_tag(i)$ we denote the tag present at the current (unconsumed) position of $IL(i)$, and we use $top\_tf(i)$ as a short notation for the tag's frequency associated with this tag.

By $unseen\_users(t,i)$ we denote the maximum number of yet unvisited users who may have tagged item i with t . This is initially set to the maximum possible tag's frequency of i over all tags (a value that is available at the current position of the inverted list of $IL(i)$, as $top\_tf(i)$).
Each time we visit a user v who tagged item i with t, we *(i)* update $sf(t|u,i)$ (initially set to 0) by adding to it $\sigma^+(u,v)$, and *(ii)* decrement $unseen\_users(t,i)$. We obtain the final social frequency value $sf(t|u,i)$ when $unseen\_users(t,i)$ reaches 0.

Lines 18 to 19 are an important part of the algorithm. We avoid expensive and hardly updatable pre-computations of the proximity values by the relaxation (see Section 3.1). Thus the proximity computation in the social network is computed on-the-fly.

In the rest of this section, we detail the STRec algorithm

Algorithm 2: Social Process

Data: $(u,i) \in U \times I$, the user u whom we want recommend tags for the item i; $v \in U$, current (direct or indirect) neighbor while scanning the social network

```
1  forall the tags t ∈ T(v,i) do
2  |  sf(t|u,i) ← sf(t|u,i) + σ+(u,v)
3  |  if t ∉ D then
4  |  |  add t to D
5  |  |  unseen_users(t,i) ← top_tf(i) /*initialization*/
6  |  end
7  |  unseen_users(t,i) ← unseen_users(t,i) − 1
8  end
9  while IL(i) ≠ ∅ AND (t ← top_tag(i)) ∈ D do
10 |  tf(t,i) ← top_t f(i) /*t's frequency in i is now known*/
11 |  advance IL(i) one position
12 |  Δ ← tf(t,i) − top_tf(i)
13 |  forall the tags t' ∈ D/CIL(i) do
14 |  |  unseen_users(t',i) ← unseen_users(t',i) − Δ
15 |  end
16 |  add t to CIL(i)
17 end
18 forall the users v' s.t. (v,v') ∈ E do
19 |  RELAX(u,v')
20 end
```

(Algorithm 3). First, let us remind that we maintain a max-priority queue H whose top element $top(H)$ will be at any moment the most relevant unvisited user on the network. At each step in the network, the top of the queue is extracted (visited) and its unvisited neighbors (adjacent nodes) are added to the queue (if not already present) and then relaxed (Algorithm 1). However, at any time of the running of the algorithm, an *optimistic* overall score $MaxScore(t|u,i)$ of any tag t that has already been seen in D is estimated as:

$$(1-\alpha) \times (top(H) \times unseen\_users(t,i) + sf(t|u,i)) + \alpha \times \max(tf(t,i), top\_tf(i))$$

Symmetrically, we estimate $MinScore(t|u,i)$, a *pessimistic* overall score, as:

$$(1-\alpha) \times sf(t|u,i) + \alpha \times \max(tf(t,i), partial\_tf(t))$$

where $partial\_tf$ represents the count of visited users who tagged i with t, which is used as a lower-bound for $tf(t,i)$ when it is not yet known. The list of candidate tags D is sorted in descending order by this lowest possible score.
An upper-bound score, $MaxScoreUnseen$, for the unseen tags is also estimated using the following value as overall frequency for each tag t:

$$\alpha \times top\_tf(i) + (1-\alpha) \times top(H) \times top\_tf(i)$$

The running of the algorithm terminates when this upper-bound score and the maximal optimistic score of tags, that

are already in D but not in its top-k, are less than the pessimistic score of the last element in the current top-k of D (i.e., $D[k]$). This is because we have the guarantee that the top-k can will no longer change as it is explained in [6].

Furthermore at each iteration, the algorithm can alternate (by calling the $CHOOSEBRANCH()$ method we describe below) between two possible execution branches: the social branch (Algorithm 2) and the textual branch, which is a direct adaptation of the Non Random Access (NRA) algorithm [14, 24].

Algorithm 3: STRec algorithm

Data: $u \in U$, $i \in I$: the user u whom we want recommend tags for the item i

forall the $(v, i, t) \in S$ **do**
  $\sigma^+(u, v) \leftarrow -\infty$
  $sf(t|u, i) \leftarrow 0$
  set $IL(i)$ position on first entry; $CIL(i) \leftarrow \emptyset$
end
$\sigma^+(u, u) \leftarrow 0$; $D \leftarrow \emptyset$ /*candidate tags*/
$H \leftarrow$ max-priority queue of nodes u (sorted by $\sigma^+(u, v)$), initialized with u
while $H \neq \emptyset$ **do**
  CHOOSEBRANCH()
  if *social branch* **then**
    $v \leftarrow EXTRACT\_MAX(H)$
    $SOCIAL\_PROCESS(u, i, v)$
  else
    if $IL(i) \neq \emptyset$ **then**
      $t \leftarrow top\_tag(i)$
      if $t \notin D$ **then**
        add t to D and $CIL(i)$
      end
      $tf(t, i) \leftarrow top\_tf(i)$
      advance $IL(i)$ one position
    else
      break
    end
  end
  if $MinScore(D[k]|u, i) > \max_{l>k}(MaxScore(D[l]|u, i))$ AND $MinScore(D[k]|u, i) > MaxScoreUnseen$ **then**
    break
  end
end
Return $D[1], \ldots, D[k]$

The $CHOOSEBRANCH()$ method considers the tag t', which has the highest potential score, and we choose the branch that is the most likely to refine the score of t'.

$$t' = D[argmax_{l>k}(MaxScore(D[l]|u, i)]$$

We set $MaxTextual$ to $\alpha \times top\_tf(i)$ if the tag's frequency $tf(t', i)$ is not yet known, and to 0 otherwise. For the social part of the score, we set $MaxSocial = (1 - \alpha) \times unseen\_users(t', i) \times top(H)$. Then, we follow the social branch if $MaxSocial$ is greater than $MaxTextual$.

The result re-ordering step

STRec returns a list D of candidate tags sorted in descending order of their scores. The k first tags of this list ($D[1], \ldots, D[k]$) are intended to be recommended to the user. In order to improve the quality of the recommendation, we add a re-ordering step that recomputes the scores of the tags in D according to degree their association degree with the best ranked tag in D. The intuition is that the first tags in the top-k result are the most relevant ones, so the scores of the lower ranked tags have to be updated according to their degree of association with the higher ranked ones. This recovery step improves the consistency and the accuracy of the result.

Fixing the first tag $t = D[1]$, we compute the confidence scores (with regard to $D[1]$) $conf(t \rightarrow t')$ of all the lower ranked tags t' in D. Then, we sort D again with obtained new score values $(1 + conf(t \rightarrow t')) \times score(t'|u, i)$. Thus, we model to some extent the interest to put in the result a tag t' in addition to the first one $D[1]$. The confidence score is computed by analysing the association degree between t and t', i.e. the number of shared items (i.e. items tagged by t and t') and the number of shared users (i.e. users who used both tags t and t'). An in depth description of the re-ordering technique and its interest is given in [7]. In the following experiments, we denote by *STRec++* the approach consisting in running the STRec algorithm with a re-ordering phase. This can improve noticeably the quality of recommendation as shown below.

4. EXPERIMENTATIONS

4.1 Datasets

We chose five datasets from four online systems: del.icio.us[2], Movielens[3], Last.fm[4], and BibSonomy[5].

We take the ones of del.icio.us, movielens, and last.fm from *HetRec 2011* [4] and the two other ones from Bibsonomy: a post-core at level 5 and a one at level 2 [3, 12]. We call them respectively *Bibson*5 and *dc*09).

*dc*09 is the one of the task 2 of ECML PKDD Discovery Challenge 2009[6]. This task was especially intended for methods relying on a graph structure of the training data only. The user, item, and tags of each post in the test data are all contained in the training data's, a post-core at level 2.Let us remaind that a post-core at level p is a subset of a folksonomy with the property, that *each user, tag and item has/occurs in at least p times.*Table 1 presents the caracteristics of these datasets.

4.2 Evaluation Measures and Methodology

To evaluate STRec, we used a variant of the leave-one-out hold-out estimation called LeavePostOut [12, 16]. In all datasets except *dc*09, we picked randomly, for each user u, one item i, which she had tagged before. Thus we create a test set and a training one. The task of our recommender was then to predict the tags the user will assign to i. We denote them $\hat{T}(u, i)$.

[2]http://www.delicious.com
[3]http://www.grouplens.org
[4]http://www.lastfm.com
[5]http://www.bibsonomy.org
[6]http://www.kde.cs.uni-kassel.de/ws/dc09/

Table 1: Caracteristics of the datasets

dataset	$\|U\|$	$\|I\|$	$\|T\|$	$\|T(u,i)\|$
Bibson5	116	361	412	2,526
dc09	1,185	22,389	13,276	64,406
del.icio.us	1,867	69,226	53,388	104,799
Last.fm	1,892	17,632	11,946	71,065
Movielens	2,113	10,197	13,222	27,713

Moreover we generate, for each training set, three social networks by computing respectively the Dice coefficient of common users' tagged items (tagged by any two users), the one of common users' tags (similar vocabulary), and the one of common users' tuples of (tag, item). We notice that we fixed the parameter α to 0.05 for all the experimentations, which is of course not necessarily optimal for all of them. We kept this value after a calibration we made on the dataset dc09. It may be seem rather small but it is comprehensible when we compare the two components of STRec. Indeed, the weights of the edges (i.e. proximity measures) between the users are in the interval [1, 0] and exceed rarely 0.5. For the performance evaluation, we use the F1-measure which is a reference in such scenarios [15, 16]. Thus for each post (u, i) in the test set, we compute the precision and recall of the top-5 recommendations as follows

$$precision(u,i) = \frac{\left|T(u,i) \cap \hat{T}(u,i)\right|}{\left|\hat{T}(u,i)\right|} \tag{7}$$

$$recall(u,i) = \frac{\left|T(u,i) \cap \hat{T}(u,i)\right|}{|T(u,i)|} \tag{8}$$

For each dataset, we average these values over all (u, i) in the test set:

$$precision = \frac{1}{|(u,i)|} \sum_{(u,i)} precision(u,i) \tag{9}$$

$$recall = \frac{1}{|(u,i)|} \sum_{(u,i)} recall(u,i) \tag{10}$$

and compute the F1-measure value as follows

$$F1 = \frac{2 \times precision \times recall}{precision + recall} \tag{11}$$

This process was repeated ten times for each dataset (except *dc*09), each time with another item and the same user, to further minimize the variance. In the sequel, the listed F1-measure values are thus always the averages over all ten runs.

4.3 Results

4.3.1 Comparison with the results of the Task 2 of ECML PKDD Discovery Challenge 2009

The Task 2 of the ECML PKDD Discovery Challenge 2009 was especially intended for the methods relying on the graph structure of the training data. The user, item, and tags of each post in the test data are all contained in the training data's post-core at level 2. There were 21 participants to this task. Table 2 shows the scores of our approach in this task, compared to the scores of the other participants. STRec reaches the eleventh place with a score of 0.30566. When we apply the re-ranking step (STRec++), with adaptive recommendations length as the others in the challenge, we improve noticeably our score up to be at the fourth place in the final result. This corresponds of an improvement of 5.53%, which confirms the efficiency of STRec++. The next experiment shows the benefits we gained on the five datasets.

Table 2: Task 2 of ECML PKDD Discovery Challenge 2009

Rank	F1-measure
1	0.35594
2	0.33185
3	0.32461
STRec++	**0.32259**
4	0.32230
5	0.32039
6	0.31396
7	0.31368
8	0.30751
9	0.30651
10	0.30566
STRec	**0.30510**

4.3.2 Benefit of sorting candidate tags relatively to the first of them

As we mentioned in Section 3.2, STRec returns a list of tags sorted by their lowest possible scores $MinScore(t|u,i)$. The top-k of this list is the recommendation result that is given to the user. STRec++ adds a re-ranking step that ensures some consistency of the vocabulary (i.e. tags' co-occurence). Table 3 shows the improvements brought by STRec++ on all the datasets we used.

Table 3: Benefit of the STRec++ approach

Dataset	F1-measure		Gain
	STRec	STRec++	
del.icio.us	0.103	0.108	5.64%
dc09	0.305	0.322	5.53%
Movielens	0,146	0.148	1.57%
Bibson5	0.389	0.397	2.00%
Last.fm	0.274	0.277	1.15%

We see that the re-ordering phase improves up to 5.6% the F1-measure of STRec for the datasets *dc*09 and *del.icio.us*. The average benefit is 3.17%. But as one can notice, the importance of the gain obtained by STRec++ varies from one dataset to another. This is due to the fact that the confidence scores we compute in the re-ordering phase depend on the number of co-occurrences of the tags (co-occurrence with the highest ranked tag). Therefore, when the users have in average a small number of posts, they share a few number of common tagged items which leads to low confidence scores.

Table 4 below confirms this analysis. It shows that for the

datasets where the user's average number of posts is greater than 50 (i.e. *del.icio.us* and *dc*09), the gain obtained by STRec++ exceeds 5%. But, this gain remains slight when the user's average number of posts is small (e.g., less than 40).

Table 4: Connection between the number of posts and the efficiency of STRec++

Dataset	#posts/user	Gain of STRec++
del.icio.us	56.13	5.64%
dc09	54.35	5.53%
lastfm	37.56	1.15%
bibsonomy	21.77	2.00%
movielens	13.11	1.57%

4.3.3 Contribution of the bounded search on the computation time

In Section 3.2, we talked about the estimation of an *optimistic* overall score $MaxScore(t|u,i)$ of a tag t that has already been seen in D and its *pessimistic* score $MinScore(t|u,i)$. We also introduced an upper-bound score, $MaxScoreUnseen$, on the yet unseen tags. This upper-bound score allows us to determine if an unseen tag may be in the top-k. Our algorithm terminates when this upper-bound score and the maximal optimistic score of tags that are already in D, but not in its top-k, are less than the pessimistic score of the last element in the current top-k of D (i.e., $D[k]$). Thus, we are guaranteed that the top-k can no longer change as exposed in [14, 6].

We measured the contribution of this optimisation which allows us to limit the search space. For this, we executed STRec without it. In other words, we search and compute the score of all the tags then we make the top-k. We call this variant STRec_*unbounded*. Table 5 lists the gains in terms of execution time obtained by the bounded search on different datasets. As one can see, the gain is significant.

Table 5: Contribution of the bounded search

Dataset	Execution time in seconds		Gain
	STRec	STRec_*unbounded*	
bibsonomy	1.4	2	30.0%
movielens	3213	4639	30.73%
delicious	2408	3379	28.72%
lastfm	4501	6619	31.99%
dc09	229	294	22.10%

For instance, it reaches 35 minutes on the dataset $lastfm$, where the unbounded search takes 1 hour and 50 minutes.

4.3.4 A weakness of STRec: the non consideration of user's tag frequency

As Table 5 shows, the STRec algorithm, as the initial search algorithms of [14, 1] we adapted to tag recommendation, is very efficient in terms of computation time. However, it does not consider the tagging behaviour of the user herself (to whom the recommendations are intended). In other words, it takes into account the tag frequency of her neighborhood but not her own tag frequency.
To evaluate the importance of user's tag frequency, we compared STRec with three baseline tag recommenders. Our first baseline recommender, *userPT*, computes all user's tag frequencies then recommends the most frequent tags. The second one, *itemPT*, uses item's tag frequencies instead of user's tag frequencies. Finally the last baseline recommender, *userItemPT*, computes a linear no-weighted combination of the two previous frequencies.

On Table 6, we can see that STRec outperforms the two first baselines. The network-dependant component of STRec clearly brings some advantages comparing to *itemPT* which is limited to item's tag frequency. But, we see that STRec fails to obtain better results than the combined frequencies *userItemPT*, except on the *dc*09 dataset. So, we can conclude that user's tag frequency play a role in this difference.

Table 6: F1-measure comparison with baselines

Dataset	STRec	userPT	itemPT	userItemPT
bibsonomy	0.389	0.296	0.373	0.466
movielens	0.146	0.093	0.132	0.170
delicious	0.103	0.187	0.101	0.186
lastfm	0.274	0.217	0.265	0.311
dc09	0.305	0.098	0.286	0.304

4.3.5 Importance of the network similarity measure

In the previous experiments, the social network is based on a similarity measure between the users. In the following, we try to analyse the impact of the network type (kind of similarity we consider) on the quality of the recommendations. Thus, we used for each training set three different social networks ($SN_{item}, SN_{tag}, SN_{(item,tag)}$), based respectively on *(i)* the Dice coefficient of common items (same item tagged by two users), *(ii)* common tags (similar vocabulary between two users), and *(iii)* common couple (tag, item) between the users.

Table 7: F1-measures according to the considered similarity

Dataset	SN_{item}	$SN_{(item,tag)}$	SN_{tag}
Bibson5	**0.39489**	0.35614	0.35529
dc09	**0.32259**	0.22842	0.23390
Last.fm	**0.27707**	0.26443	0.12939
Movielens	**0.14825**	0.12883	0.10553
del.icio.us	0.10760	**0.11646**	0.11522

Except the case of *del.icio.us* dataset, the networks built on users' common tagged items give the best results. In contrast, the networks relying on common tags give the worst results due to probably the fact that the users do not share enough vocabularies (i.e. tags).

5. RELATED WORK

Many researchers have investigated graph-based tag recommenders which rely on links between user, items, and/or tags, to make recommendations [20, 8, 18, 22, 25, 19].

We can cite the approach of Si et al. [22] which combines two already existing methods, the "most popular tags" method and FolkRank. The "most popular tags" is the simplest collaborative-filtering based technique, it recommends the most popular tags used by other users. FolkRank uses a user-item-tag tripartite hypergraph, which was first proposed as tag suggestion method in [11, 10]. Although our "tag's frequency" enables determining the most popular tags, STRec uses a simple user-graph of similarities.

In [25], Zhang et al. also combine the FolkRank algorithm with a collaborative filtering technique which considers, like us, users' similarities computed using a Pearson correlation-oriented method. Mrosek et al. [19] combine three weighted recommenders ("Tag by Source", "Tag by User", and "Tag by User Similarity"). The first one, "Tag by Source", generates tags based on the item information. It computes and uses the frequency of association of each tag to the resource. The second algorithm recommends and scores the tags used by the user. The last algorithm focuses on tags which have been used by similar users. The similarity between two users is defined over the number of same item posted by them. Note that, unlike all the cited approaches, STRec takes into account the similarity between users whether they are directly or indirectly connected (extended proximity, section 3). This allows a broader consideration of the user's neighbourhood in the social network.

Association rules mining can be used to extract from the folksonomy useful knowledge on the way users assign tags to items [21, 2], and recommendations can be done based on the extracted knowledge. In [13], Lipczak focuses on content-based tag recommenders. Hi extracts basic tags from the content of the items (e.g. the item title). Then, he extends the set of potential recommendations by related tags, proposed by a lexicon based on the co-occurrences of tags within item's posts. He determines these co-occurrences using an association rules mining technique. On their side, Wang et al. first apply the TF-IDF algorithm on the description of the item content, in order to extract from it a list of keywords [23]. Based on these keywords they use association rules to determine the most probable tags to recommend. In addition, if the item has been tagged before by other users, or if the user has tagged other items before, the history records is also exploited to detrmine the most appropriate recommendations. These algorithms are content-based, which is not the case of STRec++ where the mining step (re-ordering of the tags) is only based on the primary list of candidate tags. The association rules we use in this step take into account the association degree between the best ranked tag and its successors.

6. CONCLUSION

In this paper, we presented STRec an algorithm for tag recommendation, and an optimized variant of the algorithm STRec++ that improves the quality of the recommendations. STRec transposes the search algorithm proposed in [14, 1] to tag recommendation. One of the benefits of this algorithm is its ability to browse on-the-fly the social network of a user, which enables us to take into account the tagging behavior of the neighbourhood (direct or indirect links) of the user in the recommendation process. STRec++ improves the recommendations by applying a mining step on top of STRec that refines the final ranking of the recommended tags. This step leads to significant improvement of the quality of the recommendations as we show in the experiments.

7. REFERENCES

[1] T. Abdessalem, B. Cautis, and S. Maniu. Algorithme top-k pour la recherche dinformation dans les réseaux sociaux. In *27-èmes journées Bases de Données Avancées (BDA'11)*, Rabat, Maroc, Oct. 2011.

[2] F. M. Belém, E. F. Martins, J. M. Almeida, M. A. Gonçalves, and G. L. Pappa. Exploiting co-occurrence and information quality metrics to recommend tags in web 2.0 applications. In *Proceedings of the 19th ACM international conference on Information and knowledge management*, CIKM '10, pages 1793–1796, New York, NY, USA, 2010. ACM.

[3] D. Benz, A. Hotho, R. Jäschke, B. Krause, F. Mitzlaff, C. Schmitz, and G. Stumme. The social bookmark and publication management system BibSonomy. *The VLDB Journal*, 19(6):849–875, Dec. 2010.

[4] I. Cantador, P. Brusilovsky, and T. Kuflik. 2nd workshop on information heterogeneity and fusion in recommender systems (hetrec 2011). In *Proceedings of the 5th ACM conference on Recommender systems*, RecSys 2011, New York, NY, USA, 2011. ACM.

[5] E. W. Dijkstra. A Note on Two Problems in Connection with Graphs. *Numerical Mathematics*, 1:269–271, 1959.

[6] R. Fagin, A. Lotem, and M. Naor. Optimal aggregation algorithms for middleware. In *Proceedings of the twentieth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, PODS '01, pages 102–113, New York, NY, USA, 2001. ACM.

[7] M. Gueye, T. Abdessalem, and H. Naacke. Foldcons: A simple way to improve tag recommendation. In *Proceedings of the 5th ACM RecSys workshop on Recommender systems and the social web*, RSWeb '13, 2013.

[8] M. Gupta, R. Li, Z. Yin, and J. Han. Survey on social tagging techniques. *SIGKDD Explor. Newsl.*, 12(1):58–72, Nov. 2010.

[9] S. Hamouda and N. M. Wanas. Put-tag: personalized user-centric tag recommendation for social bookmarking systems. *Social network analysis and mining*, 1(4):377–385, 2011.

[10] A. Hotho, R. Jäschke, C. Schmitz, and G. Stumme. Folkrank: A ranking algorithm for folksonomies. In *Proc. FGIR 2006*, 2006.

[11] R. Jäschke, L. Marinho, A. Hotho, L. Schmidt-Thieme, and G. Stumme. Tag recommendations in folksonomies. In *Proceedings of the 11th European conference on Principles and Practice of Knowledge Discovery in Databases*, PKDD 2007, pages 506–514, Berlin, Heidelberg, 2007. Springer-Verlag.

[12] R. Jäschke, L. Marinho, A. Hotho, L. Schmidt-Thieme, and G. Stumme. Tag recommendations in social bookmarking systems. *AI Commun.*, 21(4):231–247, Dec. 2008.

[13] M. Lipczak. Tag recommendation for folksonomies oriented towards individual users. *ECML PKDD discovery challenge*, 84, 2008.

[14] S. Maniu, B. Cautis, and T. Abdessalem. Efficient top-k retrieval in online social tagging networks. *CoRR*, abs/1104.1605, 2012.

[15] L. B. Marinho, A. Nanopoulos, L. Schmidt-Thieme, R. Jäschke, A. Hotho, G. Stumme, and P. Symeonidis. Social tagging recommender systems. In Ricci et al. [20], pages 615–644.

[16] L. B. Marinho and L. Schmidt-Thieme. Collaborative tag

recommendations. In C. Preisach, H. Burkhardt, L. Schmidt-Thieme, and R. Decker, editors, *GfKl*, Studies in Classification, Data Analysis, and Knowledge Organization, pages 533–540. Springer, 2007.

[17] A. K. Milicevic, A. Nanopoulos, and M. Ivanovic. Social tagging in recommender systems: a survey of the state-of-the-art and possible extensions. *Artif. Intell. Rev.*, 33(3):187–209, Mar. 2010.

[18] E. Milios. Corpus-based term relatedness graphs in tag recommendation. In *Proceedings of the 23rd Canadian conference on Advances in Artificial Intelligence*, AI'10, pages 3–3, Berlin, Heidelberg, 2010. Springer-Verlag.

[19] J. Mrosek, S. Bussmann, H. Albers, K. Posdziech, B. Hengefeld, N. Opperman, S. Robert, and G. Spira. Content- and graph-based tag recommendation: Two variations. In F. Eisterlehner, A. Hotho, and R. Jäschke, editors, *ECML PKDD Discovery Challenge 2009 (DC09)*, volume 497, pages 189–199, Bled, Slovenia, September 2009. CEUR Workshop Proceedings.

[20] F. Ricci, L. Rokach, B. Shapira, and P. B. Kantor, editors. *Recommender Systems Handbook*. Springer, 2011.

[21] C. Schmitz, A. Hotho, R. Jäschke, and G. Stumme. Mining association rules in folksonomies. pages 261–270. 2006.

[22] X. Si, Z. Liu, P. Li, Q. Jiang, and M. Sun. Content-based and graph-based tag suggestion. In F. Eisterlehner, A. Hotho, and R. Jäschke, editors, *ECML PKDD Discovery Challenge 2009 (DC09)*, volume 497, pages 243–260, Bled, Slovenia, September 2009. CEUR Workshop Proceedings.

[23] J. Wang, L. Hong, and B. D. Davison. Rsdc '09: Tag recommendation using keywords and association rules. In F. Eisterlehner, A. Hotho, and R. Jäschke, editors, *ECML PKDD Discovery Challenge 2009 (DC09)*, volume 497, pages 261–274, Bled, Slovenia, September 2009. CEUR Workshop Proceedings.

[24] S. A. Yahia, M. Benedikt, L. V. S. Lakshmanan, and J. Stoyanovich. Efficient network aware search in collaborative tagging sites. *Proc. VLDB Endow.*, 1(1):710–721, Aug. 2008.

[25] Y. Zhang, N. Zhang, and J. Tang. A collaborative filtering tag recommendation system based on graph. In F. Eisterlehner, A. Hotho, and R. Jäschke, editors, *ECML PKDD Discovery Challenge 2009 (DC09)*, volume 497, pages 297–306, Bled, Slovenia, September 2009. CEUR Workshop Proceedings.