Network-Aware Search in Collaborative Tagging Applications: Instance Optimality versus Efficiency

Silviu Maniu, Bogdan Cautis

University of Hong Kong & Univ. Paris-Sud / INRIA Saclay

CIKM 2013

A Web of social interactions

Social Web: new development to the Web – users, their relationship and their data.

Significant and highly qualitative portion of the Web:

- either built as explicitly social (Facebook, Google+, Twitter), or
- content-based, but with social communities acting as "engine" (Wikipedia, blogs, forums) - implicitly social

Larger user bases $\rightsquigarrow$ better search models for data relevance, freshness guarantees.

The social tagging context

Collaborative tagging networks: a good abstraction of social Web applications

- users form a social network (may reflect proximity, similarity, friendship, closeness, etc)
- user tag items (e.g.,documents, URLs, photos, etc) from a public pool of items
  - examples: Flickr, Delicious, Netflix, Youtube, even Twitter

Users search for items having certain tags

Outline

Problem definition

Solution

Approximation algorithms

Experiments

Outline

The social tagging context

Setting: a collaborative tagging environment in which we have:

- set of items I, set of users U, set of tags T.
- *tagging* relation: `Tagged(user,item,tag)`

Social Network: undirected weighted graph $G = (\mathcal{U}, E, \sigma)$

- nodes represent users
- σ associates to each edge $e = (u_1, u_2)$ a value in $(0, 1]$ called the proximity between u_1 and u_2 (a social score)

The top-k retrieval problem

Problem (Top-k retrieval)

Given a seeker s, a query $Q = \{t_1, ..., t_r\}$ (a set of r distinct tags), an integer value k, and an item scoring model, retrieve the top-k items

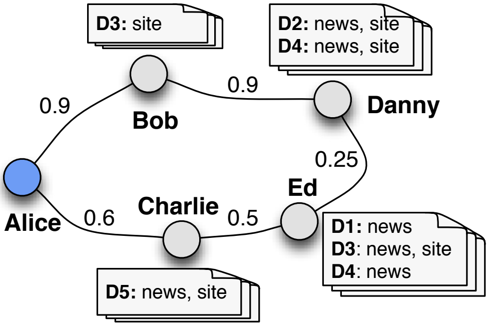
- main challenge: efficiency and applicability

Difference to classical search: item scores depend on its taggers and their proximity to the seeker

Search example

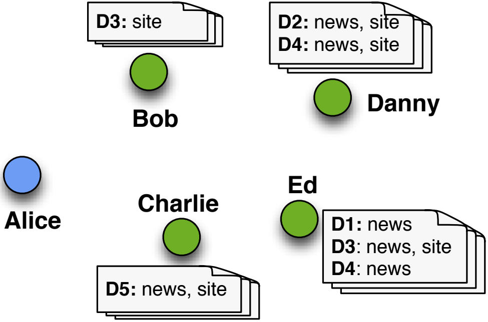
Scenario: Alice wants top-2 documents for query $\{news, site\}$

$\rightsquigarrow$ result we desire is **D4, D2**



What would happen in classical search?

each tag has same weight in **term-frequency** lists; TA/NRA [Fagin01]

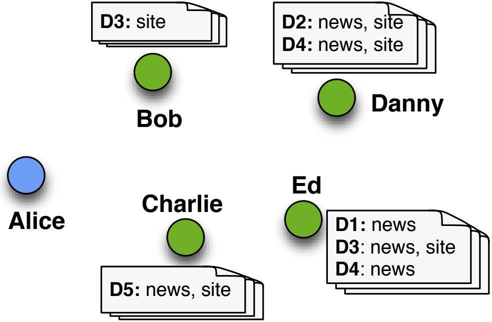


news doc	tf	*site* doc	tf
D4	2.00	D3	2.00
D3	1.00	D4	1.00
D5	1.00	D5	1.00
D2	1.00	D2	1.00
D1	1.00	D1	0.00

What would happen in classical search?

each tag has same weight in **term-frequency** lists; TA/NRA [Fagin01]

$\rightsquigarrow$ top-2 result: **D4**, **D3**



news		*site*	
doc	tf	doc	tf
D4	2.00	D3	2.00
D3	1.00	D4	1.00
D5	1.00	D5	1.00
D2	1.00	D2	1.00
D1	1.00	D1	0.00

Social frequency

Classic term frequency is replaced by a monotonic measure depending on the seeker and a parameter α:

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

$sf(i \mid u, t)$ represents the social frequency:

$$sf(i \mid u, t) = \sum_{v \in \{v \mid Tagged(v,i,t)\}} \sigma(u, v)$$

The extreme cases:

- $\alpha = 1$ classical search
- $\alpha = 0$ exclusively social search

Social frequency

Classic term frequency is replaced by a measure depending on the seeker and a parameter α:

$$fr(i \mid u, t) = \qquad\qquad\qquad\qquad sf(i \mid u, t)$$

$sf(i \mid u, t)$ represents the social frequency:

$$sf(i \mid u, t) = \sum_{v \in \{v \mid Tagged(v,i,t)\}} \sigma(u, v)$$

The extreme cases:

- $\alpha = 1$ classical search
- $\alpha = 0$ exclusively social search

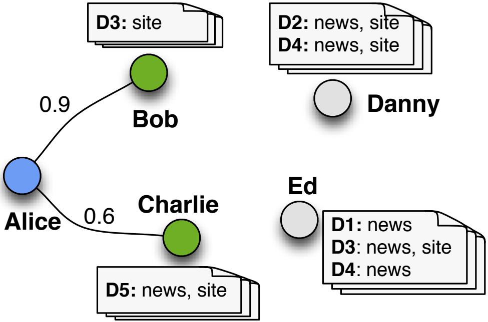
The scoring model

$score(i \mid u, t)$ is the score of item i for the given seeker u and tag t: e.g., *tf-idf*: $score(i \mid u, t) = fr(i \mid u, t) \times idf(t)$, BM25

$score(i \mid u, Q)$ is the overall score of i for seeker u and query Q: e.g., monotone aggregation functions: `sum, max, avg`

Considering only direct connections

scores depend on proximity: **per-seeker** lists [Amer-Yahia08]

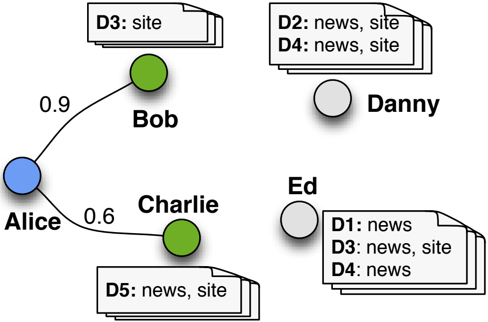


user	prox.
Bob	0.90
Charlie	0.60
Danny	0.00
Ed	0.00

Considering only direct connections

scores depend on proximity: **per-seeker** lists [Amer-Yahia08]

⇝ top-2 result: **D5**, **D3**



news		*site*	
doc	tf	doc	tf
D5	0.60	D3	0.90
D3	0.00	D5	0.60
D4	0.00	D4	0.00
D2	0.00	D2	0.00
D1	0.00	D1	0.00

user	**prox.**
Bob	0.90
Charlie	0.60
Danny	0.00
Ed	0.00

Outline

Problem definition

Solution

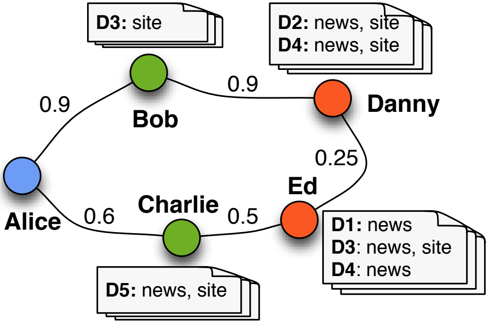
Approximation algorithms

Experiments

Adding indirect connections

indirectly connected users should be relevant too

e.g., Danny might be more similar to Alice than Charlie



user	prox.
Bob	0.90
Charlie	0.60
Danny	**?**
Ed	**?**

Extended proximity

Extended proximity (σ^+): we want to compute a social score even for users that are not directly connected to s.

$$sf(i \mid u, t) = \sum_{v \in \{v \mid Tagged(v,i,t)\}} \sigma^+(u, v)$$

Possible definitions for σ^+

On a path, $p = (u_1, \ldots, u_l)$, monotonically aggregate the weights:

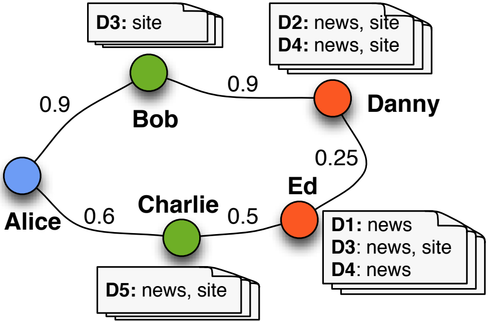
- *Example 1:* path multiplication $\sigma^+(p) = \prod_i \sigma(u_i, u_{i+1})$
- *Example 2:* minimum value on a path $\sigma^+(p) = min\{\sigma(u_i, u_{i+1})\}$
- *Example 3:* exponential decay $\sigma^+(p) = \lambda^{-\sum_i \frac{1}{\sigma(u_i, u_{i+1})}}$, $\lambda \geq 1$

Then choose the optimal of the aggregated paths:

$$\sigma^+(s, u) = max_p\{\sigma^+(p) \mid s \overset{p}{\rightsquigarrow} u\}.$$

Considering indirect connections

per-seeker proximity lists

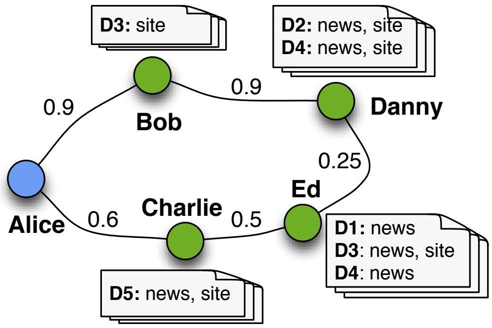


user	prox.
Bob	0.90
Danny	**0.9x0.9**
Charlie	0.60
Ed	**0.6x0.5**

Considering indirect connections

per-seeker proximity lists

⇝ top-2 results: **D4**, **D2**



news		*site*	
doc	tf	doc	tf
D4	1.11	D3	1.20
D2	0.81	D4	0.81
D5	0.60	D2	0.81
D3	0.30	D5	0.60
D1	0.30	D1	0.00

user	**prox.**
Bob	0.90
Danny	0.81
Charlie	0.60
Ed	0.30

Algorithm

Our goal: early-termination algorithm, in the style of TA/NRA: inverted lists of *tf* values, and proximity lists are acessed sequentially.

Key subroutine: getting the next closest user

- pointer increment in the user proximity lists

Algorithm

Previous approach: ContextMerge [Schenkel08] - precomputed proximities for all user pairs.

Major drawbacks:

- high disk space cost: $\sim$ 700 TB for Delicious, even bigger for Facebook
- limited applicability: social scores can evolve (e.g., tag similarity), lists need to be kept up to date

Our contribution: the SNS algorithm

Observation

The visit of the network in decreasing order of proximity (w.r.t the seeker) can be done on the fly and as needed

- for a wide family of σ^+ functions - monotone functions (including the 3 examples)

Our contribution: the SNS algorithm

Observation

The visit of the network in decreasing order of proximity (w.r.t the seeker) can be done on the fly and as needed

- for a wide family of σ^+ functions - monotone functions (including the 3 examples)

Advantages:

- a typical network can easily fit in main-memory
- spare the potentially huge disk volumes required previously
- social score updates become a non-issue
- full personalization: each seeker can choose any function

Computing the proximities

The previous three σ^+ examples satisfy the following property (Descending monotonicity):

Property

Given a social network G and a seeker user s, for any other user v in G that is connected to s we have $\sigma^+(s, v) \geq \sigma^+(s, v.previous)$.

$\rightsquigarrow$ the proximities for a given seeker can be greedily computed on the fly - by generalizing Dijkstra's algorithm.

Formal guarantees: correctness

Property

SNS *visits the users of the network in decreasing order of their* σ^+ *values with respect to the seeker.*

Corollary

SNS *visits the users who may be relevant for a query in the same order as* ContextMerge *(or equivalent algorithms) and hence outputs equivalent results.*

Formal guarantees: instance optimality

Theorem

$\mathrm{SNS}_{\alpha=0}$ is instance optimal over all algorithms that do not make "wild guesses" and over all inputs D, when $cost(A, D) = users(A, D)$.

Outline

Problem definition

Solution

Approximation algorithms

Experiments

Is instance optimality enough?

No, at least in most practical scenarios

Two main reasons:

- the search may visit a significant part of the network, yet the final top-k is established relatively soon,
- computing exact shortest paths, even in an optimal manner, still has a significant execution overhead.

Is instance optimality enough?

No, at least in most practical scenarios

Two main reasons:

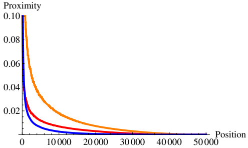
- the search may visit a significant part of the network, yet the final top-k is established relatively soon,
- computing exact shortest paths, even in an optimal manner, still has a significant execution overhead.

Possible directions:

1. *tighter bounds for the termination condition*, by estimating unseen proximities,
2. *estimate all user proximities*, via approx. shortest paths.

Approximations - estimating score bounds

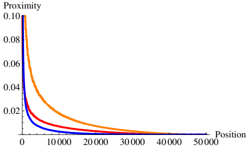
Termination conditions too weak in practice, values drop rapidly:



⇝ having a high-level description of the proximities, can lead to tighter score estimations but approximate results

Approximations - estimating score bounds

Termination conditions too weak in practice, values drop rapidly:



$\rightsquigarrow$ having a high-level description of the proximities, can lead to tighter score estimations but approximate results

Two approaches for estimation using a probabilistic parameter δ:

1. storing high level statistics, mean and variance (SNS/MV)
2. storing histograms of the proximity vectors (SNS/H)

Approximations - estimating user proximities $\mathrm{SNS/L}$

Adaptation of the landmarks approach of [Potamias09]:

- for a number of d landmarks, compute the entire proximity vector
- by triangle inequality, we can obtain upper and lower bounds of any seeker, user pair:

$$\min\left(\frac{\sigma^+(s,l_i)}{\sigma^+(l_i,v)}, \frac{\sigma^+(l_i,v)}{\sigma^+(s,l_i)}\right) \geq \boldsymbol{\sigma}^+(\mathbf{s},\mathbf{v}) \geq \sigma^+(s,l_i) \times \sigma^+(l_i,v)$$

- by acessing sorted landmark lists *à la* $\mathrm{TA} \rightsquigarrow$ we can estimate user proximities without needing expensive priority queues.

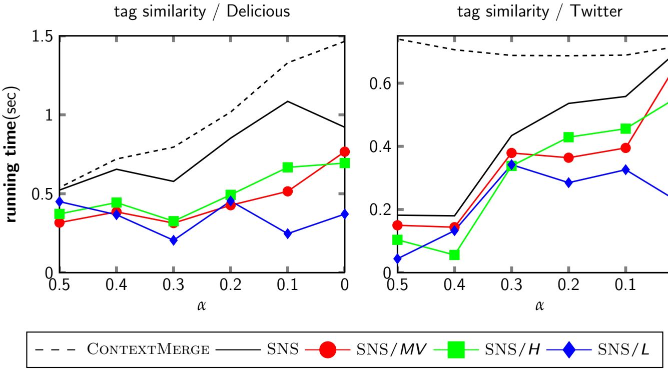
Outline

Experiments: datasets

Statistic	Delicious	Twitter
users	80,000	570,387
items	595,811	1,570,866
tags	198,080	305,361
(u, i, t) triples	2,863,365	8,753,706
distinct items/user	9.59	10.10
distinct tags/item	4.22	1.39
distinct tags/user	15.64	9.45

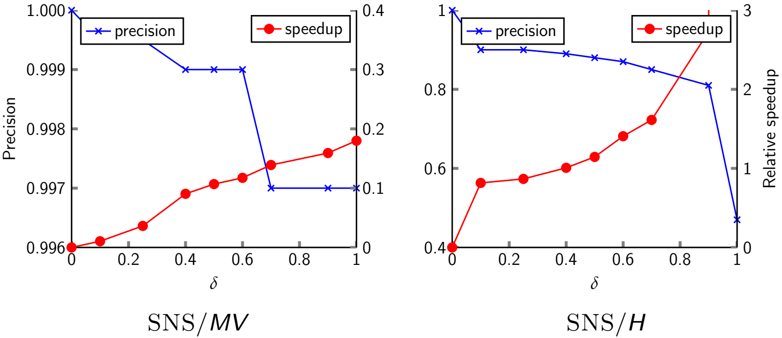
For experiments, three pairwise similarity networks (tag, item, and item-tag)

Experiments: performance



- significant gains in performance, for both exact and approximate approaches

Experiments: precision/speed tradeoff



- high precision even when using simple statistics and independence assumptions

What / when?

- exact approach: very sparse network, very low proximity values, in cases where access is limited (e.g., Web API requests), relatively low k,
- histograms/bounds approach: sparse network and low proximity values, high k,
- landmarks approach: denser networks with (relatively) high proximity values, high k.

To summarize

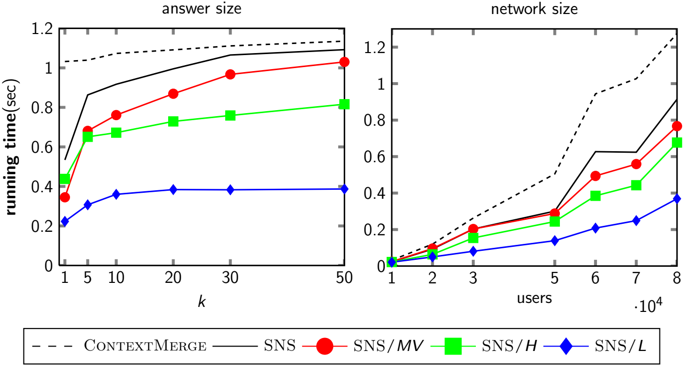
- novel algorithm generalizing shortest path based social search
- instance optimal in the exclusively social case for monotonic proximity functions,
- approximate approaches can be added to the framework, for further efficiency

Thank you.

$\mathrm{SNS}_{\alpha=0}$ (general flow)

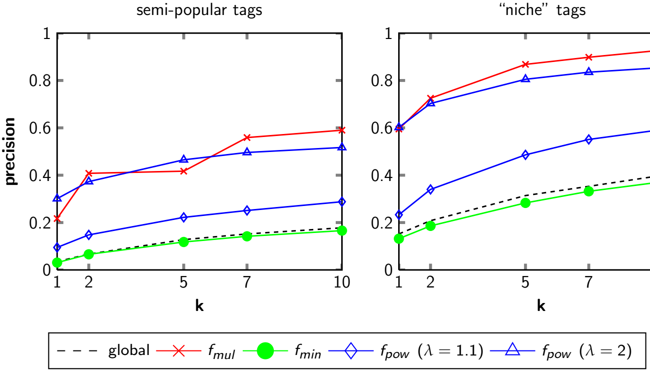
Require: seeker s, query $Q = (t_1, \ldots, t_r)$
  initialization of the distances and max-priority queue H
  candidate list $D = \emptyset$, seeker s entered to queue H
  while exists an user u in the queue H **do**
    compute $\sigma^+(s, u)$
    get all documents d belonging to u, tagged with $t \in Q$
    compute their scores and insert them into D
    prune the heads of the lists IL, removing documents in D
    refine (or relax) the proximity scores for neighbours of u from G
    if $\textsc{MinScore}(D[k], q) \geq max_{l>k}(\textsc{MaxScore}(D[l], q)$ $\textsc{and}$
      $\textsc{MinScore}(D[k], q) \geq \textsc{MaxScoreUnseen}$ **then**
      break
    end if
  end while
  return $D[1], \ldots, D[k]$

Experiments: scalability



- performance gains increase with network size, gains relatively constant with k

Experiments: relevance



- social keyword queries can provide good prediction accuracy for bookmarking