e#: Sharper Expertise Detection from Microblogs [Application Paper]

Thibault Sellam
CWI, the Netherlands
thibault.sellam@cwi.nl

Martin Hentschel*
Snowflake
hemartin@snowflake.net

Vasilis Kandylas
Microsoft
vakandyl@microsoft.com

Omar Alonso
Microsoft
omalonso@microsoft.com

ABSTRACT

Microblogging platforms such as Twitter provide low cost access to an immense reserve of authoritative professionals, opinion leaders and hobbyists for a wide range of topics. Yet, as microposts are short and incredibly diverse, many of these experts are hidden. In this paper, we present e#, a system to retrieve experts automatically for a given set of keywords. Our design targets exhaustivity: e# can detect previously undetectable experts. The core idea is to enhance a state-of-the-art expert detection algorithm with a graph of expertise domains. Our system produces this graph from hundreds of Gigabytes of Web search query logs and behavioral data, processed in a distributed, parallel fashion. We provide a detailed description of our architecture, including an original SQL-based community detection algorithm. We then benchmark our system with 750 queries, using crowdsourcing. We observe that e# finds many more experts than a state-of-the-art baseline.

Keywords

Expert detection, query expansion, clustering

1. INTRODUCTION

Microblogging offers a mighty, low-cost means to disseminate and consume knowledge. Platforms such as Twitter let political analysts comment elections, sports journalists explain why their favorite team fell short, and technology fans criticize the weight of a new phone. In this paper, we investigate the problem of *expertise detection*: we want to retrieve experts from microblogs, given a topic expressed as a set of keywords. For example, suppose that we wish to learn more about American football from Twitter. Given a set of keywords such as `49ers` or `NFL` as input, can we return a list of all the authoritative Twitter accounts?

*Martin Hentschel was affiliated with Microsoft at the time of this work

Expertise detection systems must achieve high *precision* and high *recall*. *Precision* measures the purity of the results. It is the proportion of experts returned by the system which are relevant to the topic. On Twitter, achieving high precision is challenging because the data contains an extravagantly large range of topics and vocabulary: it contains spam, fake accounts, but also many ambiguities. In our example, the simple term `football` designates a different sport in Europe and America. *Recall* measures the exhaustivity of the results. It is the proportion of relevant experts on the whole microblogging platform detected by our system. Recall is challenging because tweets are short. An expert in `49ers` is likely to be an expert in `West Coast football` too, because the 49ers is a popular football team from the US West Coast. Yet, as tweets cannot contain more than 140 characters, the chance to have both expressions in the same post is low. Therefore, a search for `49ers` may miss the experts for `West Coast football`.

Expertise detection has been studied for decades, but in a very different context: initially, it focused on finding experts from enterprise documents, in order to smoothen collaboration between employees. The corpora were small, heterogeneous and the queries were very specific (e.g., `FORTRAN developer`). With social media, the context is different: the topics of interest can be narrow (e.g., `49ers draft`) or broad (e.g., `sports`). The corpora are homogeneous (all messages have the same format), but their scale is massive. Also, the requirements in terms of precision and recall are different. In enterprise settings, the aim was to initiate professional collaborations, thus false positives were very costly. Most studies on expertise retrieval targeted precision, recall came as distant second [4]. In contrast, our users are looking for information sources, not collaborators. Hence, they value depth and variety, while false positives are relatively cheap. This shifts the balance towards recall. Unfortunately, achieving high recall is also much harder on social media, because microposts have a short length and an immense vocabulary.

How can we detect experts with both high recall and high precision? We present e#, a system to detect previously undetectable experts. Our strategy is based on *query expansion*, well known in the context of document search but seldom used for expert detection. We operate in two steps, offline and online. Offline, we build a collection of linked topics of expertise from Web data. Online, we exploit this collection to *augment* incoming queries, and feed the result to a precision-based expert detector. We obtain a variety of

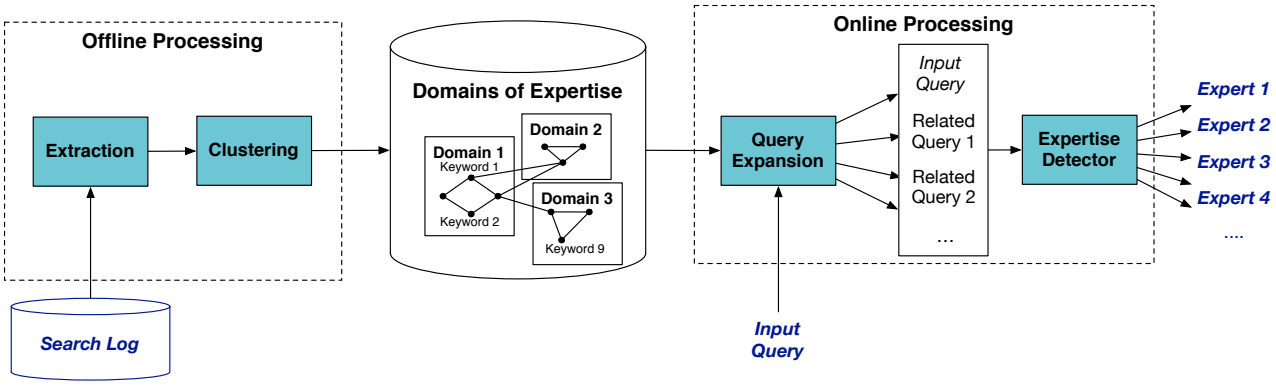


Figure 1: Overview of e# - our system augments an input query with related queries, inferred from the search log of a commercial search engine.

high quality experts.

Two questions remain. First, how do we collect and link topics of expertise? We propose to exploit the search query log of a commercial search engine. This source gives us a massive, time-relevant collection of keywords. We infer the semantic associations between the terms with search and click behavior. Second, how do we exploit this collection? Our approach is to partition the search terms into *communities*, that is, groups of strongly related keywords. We then use these groups to enrich the queries. Because of the scale of the datasets involved, the engineering effort is non trivial. We present an original implementation of *modularity maximization*, a framework to detect communities in graphs. The advantage of our approach is that it can directly be implemented in (parallel) declarative languages such as Hive, Pig, Microsoft's SCOPE or even SQL.

To summarize, here are our contributions:

- We present our pipeline e#, which combines search query log analysis, community detection and query expansion at scale.
- We introduce a parallel, distributed algorithm to infer clusters of related keywords from large Web search logs.
- We describe a complete experimental evaluation, with real-life examples and a crowdsourcing study. We present our results on 750 queries from many different topics.

The rest of this paper is organized as follows. In the following section, we give an overview of e#. We then present the base expertise detection algorithm on which we built e#. In the fourth section, we detail how e# builds collections of related keywords to expand the queries. We expose how our system matches queries and expertise domains in the fifth section. We present our experiments in the sixth section. We then describe related works and conclude.

2. OVERVIEW

The main idea behind e# is to enhance an existing expert detection algorithm with a collection of expertise domains. Figure 1 gives an overview of our pipeline. It depicts two stages: an offline stage, during which we build the collection, and an online stage, during which we exploit it.

The offline stage can itself be decomposed in two steps. First, we process a search query log. Using search terms and clicks, we build a weighted graph, in which each vertex represents a keyword and each edge represents a semantic association. Then, we detect *communities* in this graph, with a custom SQL-based algorithm. Each of the communities we obtain describes a topic of expertise, exploitable for query augmentation.

During the online stage, we run the actual query augmentation: we match the query with a topic of expertise from the database, and append the corresponding keywords. We then run a detection algorithm presented previously in the literature [14]. Section 3 presents the algorithm.

Thanks to the query log, our collection of domains is inherently current. For instance, at the time of writing, it contained keywords related to new technological products (`smart watches` or `VR glasses`) or upcoming media events (e.g., `Star Wars VII`). This is particularly useful when dealing with social media. Also, entries often come in many variants (e.g., `footbball`, `fotbal`, `foot`, etc...). This variety improves the robustness of our system at little CPU cost.

3. PRELIMINARIES - BASELINE

In this section, we present the algorithm on which we built e#. Expertise detection involves two main challenges: *candidate selection* and *expertise ranking*. Candidate selection is the problem of finding candidate experts for a given topic. Expertise ranking is the problem of determining the strength of expertise given textual evidence. To solve both problems, we use a method proposed recently by Pal and Counts [14], shown to be competitive for Twitter data. The framework was simplified for production purposes, it currently runs in a commercial environment.

We implemented candidate selection on Twitter as follows. A candidate expert is either an author of a tweet, or a person mentioned in a tweet. In both cases, the tweet must match the query. By default, a tweet matches a query if it contains all of its terms after lower-casing [4, 14].

For expertise ranking, we first compute features of textual evidence, and then rank the candidates on these features. In their paper, Pal and Counts evaluate a dozen features. We kept those which they present as important: the topical signal (TS), the mention impact (MI), and the retweet impact (RI). These features are defined as follows:

$$TS = \frac{\#tweets\ by\ user\ on\ topic}{\#tweets\ by\ user}$$
$$MI = \frac{\#mentions\ of\ user\ on\ topic}{\#mentions\ of\ user}$$
$$RI = \frac{\#retweets\ of\ user's\ tweets\ on\ topic}{\#retweets\ of\ user's\ tweets}$$

The first two feature, TS and MI, measure how much the user is specialized in the topic of interest. The third feature, RI, measures the influence of the user.

Before we perform the ranking, we normalize and aggregate the features. To normalize the features, we compute their z-score. For instance, if μ_{TS} is the average of TS and σ_{TS} its standard deviation, we compute $z_{TS} = \frac{x-\mu_{TS}}{\sigma_{TS}}$. In practice, the features appear to be log-normally distributed. Therefore, we take their logarithm to obtain Gaussian distributions. To aggregate the scores, we used a weighted sum, using the authors' guidelines.

In their paper, Pal and Counts propose an optional filtering step, based on cluster analysis. This step is computationally expensive, and it is contrary to our objective of improving recall. Therefore, we discarded it in our implementation.

4. COLLECTING TOPICS OF EXPERTISE

In this section, we describe how we build our collection of expertise domains. During the *extraction* phase, we derive a graph of semantic relationships from the search query log. During the *clustering* phase, we detail how to decompose this graph into communities, using a parallel, modularity-based approach.

4.1 Extracting Semantic Relationships

To build our collection of related topics, we exploit the search log of a commercial search engine. We chose this source because it is intrinsically current and exhaustive.

How can we infer semantic connections between terms from a search log? We propose to exploit the URLs clicked for each keyword. This approach lets us detect non-obvious semantic associations, and it is practical to implement [1]. Consider a vector space where each dimension represents a URL from the query log. In this space, we associate each query to a vector. Each component of the vector represents the number of clicks on the URL. To obtain the similarity between two terms, we can compute the cosine distance between the two vectors which represent them. If we compute the distance between every possible pair of terms, we obtain a term similarity graph. In this weighted, undirected graph, each vertex represents a query, and the edges describe their similarity. We illustrate this operation with Figure 2. This graph gives us the material for our next step: the community detection.

In practice, a few adjustments are necessary. For instance, we remove all the queries which appear less than 50 times per month, to reduce noise and save space. Even after this operation, the same term can appear with dozens, sometimes hundreds of variants (e.g.., `san francisco`, `#sanfrancisco`, `sf`, ...). We leave these queries unchanged (no stemming, or correcting), in order to capture as many different cases as possible.

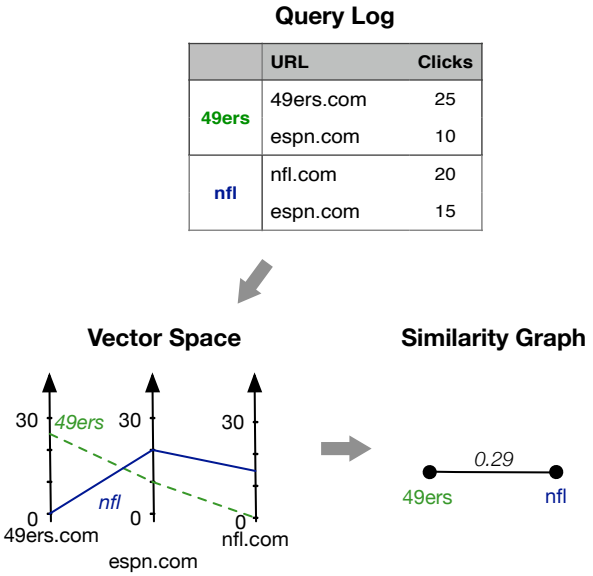


Figure 2: Extracting the similarity between terms from the search log.

4.2 Detecting High-Level Domains

Once the similarity graph is built, our next step is to create groups of related keywords. We solve this problem with *community detection*. The idea is to identify groups of queries which are densely connected to each other, but loosely connected to the rest of the graph. We assume that if a group of keywords obeys such a property, then we can use it to expand queries. The network analysis literature contains dozens of ways to formalize this notion [10]. We base our system on modularity maximization [13], which is simple and widely studied. We first present the original sequential algorithm, proposed by Newman et al., then we introduce our parallel variant.

4.2.1 Modularity Maximization

Overview. Consider an undirected graph $G = (V, E)$. For the sake of presentation, we consider that this graph is not weighted, but that more than one edge can connect two nodes[1]. Consider a set of vertices $C \subset V$. The modularity measures how densely connected C is. To compute it, we count the number of edges within the set, and compare to what we would expect if the edges were drawn randomly between G's vertices, preserving the vertex degrees. The modularity is the difference between these two terms. Let E describe the expected value:

$$Modularity = \#edges - E[\#edges] \quad (1)$$

Partition G's vertices into p partitions $C_1, \ldots, C_p$. If we sum the modularities of each of these partitions, we obtain the *total modularity*:

$$TMod(\{C_1, \ldots, C_p\}) = \sum_{i \in 1..p} Mod(C_i) \quad (2)$$

[1]We can convert the similarity graph described in the previous section into this representation. To do so, we rescale and discretize the weights to obtain integers. Then, we create one edge for each unit.

This is our objective function. A high value indicates that we found many dense communities. A low value means that either the graph does not contain any community, or that the partitioning is sub-optimal.

Computing the Modularity. We defined the modularity as the difference between the number of edges within a set of vertices and the expected number of edge within this set. To obtain the first quantity, we can simply count. Now, how do we obtain the second one?

For a set of vertices C, the variable m_C describes the number of edges and $E[m_C]$ the expected number of edges. We have:

$$Mod(C) = m_C - E[m_C] \quad (3)$$

We want to compute $E[m_C]$. Draw an edge at random between two vertices of G. Let P_C describe the probability that the edge connects two vertices of C, and let m_G denote the number of edges in the graph. We obtain:

$$E[m_C] = m_G * P_C \quad (4)$$

Let's compute the probability P_C. Let $D_G = 2 * m_G$ represent the sum of all the degrees of all the vertices of the graph, and let D_C represent the sum of all the degrees of the vertices in C. For a given edge, the probability that one of the endpoints ends up in the set C is D_C/D_G . Therefore, the probability of having both endpoints in the community is:

$$P_C = (D_C/D_G)^2 \quad (5)$$

Putting everything together, we obtain:

$$Mod(C) = m_C - m_G * (D_C/D_G)^2 \quad (6)$$

Note that the modularity is often normalized: many authors use $Mod(C)/m_G$ instead of $Mod(C)$. As m_G is a constant, this approach is equivalent to ours.

Greedy Heuristic. Maximizing modularity is a NP-hard problem [13]. The seminal single-machine heuristic, presented by Newman et al., operates in a greedy, bottom-up manner. We initialize the algorithm by assigning each vertex to its own community. Then, at each iteration, we find the two closest communities, and merge them. We stop when we cannot improve the score anymore, or when we have reached a satisfying number of communities.

The critical part of the algorithm is to define "closest". According to Newman, two communities are close if merging them leads to an improvement in the global modularity. Formally, if C_1 and C_2 describe these communities, we have:

$$\Delta Mod = Mod(C_1 \cup C_2) - Mod(C_1) - Mod(C_2) > 0 \quad (7)$$

Instead of computing the three terms of this equation separately, we can use a computational shortcut [13]. If $m_{1\leftrightarrow 2}$ represents the number of edges between C_1 and C_2, we have:

$$\Delta Mod = m_{1\leftrightarrow 2} - E[m_{1\leftrightarrow 2}] \quad (8)$$

Let D_1 and D_2 represent the sum of degrees of C_1 and C_2's vertices. We obtain the second term as follows:

$$E[m_{1\leftrightarrow 2}] = \frac{D_1 * D_2}{2 * m_G} \quad (9)$$

4.2.2 SQL-based Modularity Maximization

To deal with the scale of commercial search engine query logs, we developed a custom variant of Newman's procedure.

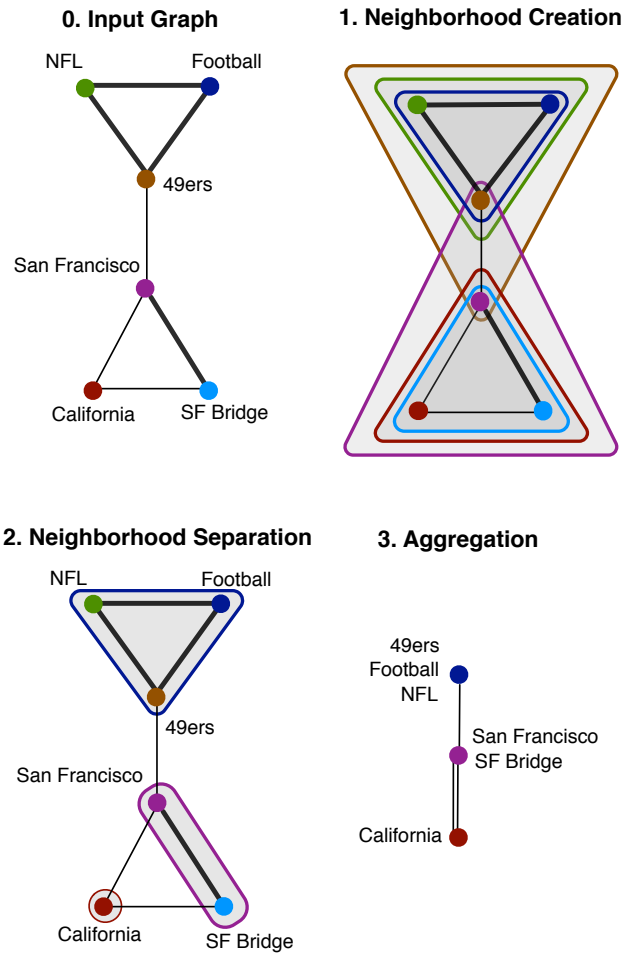


Figure 3: First iteration of our modularity optimization algorithm on a fictive example. The shaded triangles represent neighborhoods.

Compared to previously published frameworks such as [17], our approach can be directly implemented in a SQL-like language such as Hive, Microsoft's SCOPE or Pig. Therefore, we can parallelize it with standard map-reduce relational operators [5].

As previously, we initialize the algorithm by assigning each vertex to a community. Then, we repeat the following three steps:

1. For each community, list all the *neighbor* communities. Two communities are neighbors if (a) they are connected and (b) if we union them, the total modularity increases ($\Delta Mod > 0$). We obtain several *neighborhoods*, one for each community.

2. The neighborhoods found in Step 1 are overlapping: one community may belong to several neighborhoods. To remediate this, take each community, list all the neighborhoods to which it belongs and keep the closest one (ΔMod is as large as possible).

3. For each neighborhood, aggregate all the communities into one large, new community

```
neighbors = select c1.query as query1,
                   c2.query as query2,
                   distance
        from graph
        inner join communities c1 on query2
        inner join communities c2 on query1
        where ModulGain(query1,query2) > 0;

partitions = select query2,
                  argmax(distance, query1)
             from neighbors
             group by query2;

communities = select query1 as comm_name,
                     query2 as query;
```

Figure 4: Body of the community detection algorithm in pseudo-SQL. The table `Graph(query1, query2, distance)` represents the graph, and `Communities(comm_name, query)` represent the communities (the foreign key relationships are underlined).

We illustrate these steps in Figure 3, and present the pseudo-code in Figure 4.

4.2.3 *Parallelization and Optimization*

We presented our algorithm in pseudo-SQL. This approach is *declarative*: we rely on the data management system to effectively parallelize the code. We now discuss a few query processing methods to achieve good performance.

The most time consuming operation is the join between the communities and the graph, necessary to list the neighborhoods (the first query in Figure 4). If the nodes have enough main memory, we can speed it up with a replicated join: we replicate and index the `communities` table at each node. Then, we split the `graph` table, broadcast the partitions, and execute the join at each node. If this is not possible, we must chain two map-side joins. We cluster the tables `communities` and `graph` on the join keys (first `query1`, then `query2`), send each partition to a node, then perform the join at each node.

The following two operations (grouping and renaming) are much simpler, and can be executed in one map-reduce pass. The mappers emit the tuples with the key `query2`, then the reducers perform the aggregation and the renaming.

5. QUERY MATCHING

We now describe how retrieve a community for a given query. Our approach is based on exact match: we find the community which contains the query terms exactly and in order, after lower-casing. Once we identified the relevant community, we run the expert search for all the related terms separately. We then union the results and rank the experts. This approach is purposely conservative, and it is straightforward to implement.

An advantage of production query logs is that terms often come in hundreds of variants, with alternative spellings and mistakes. This improves the flexibility of the matching at little computational cost.

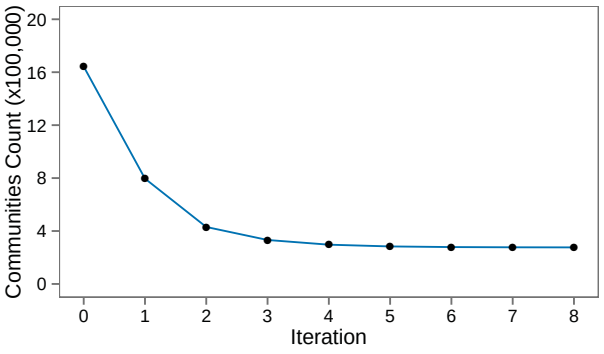


Figure 5: Convergence of the community detection algorithm.

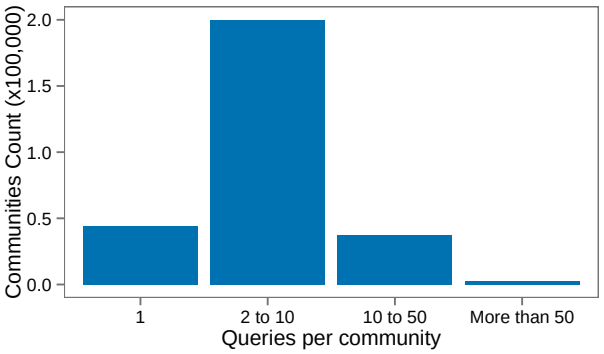


Figure 6: Distribution of the community sizes.

6. EXPERIMENTS AND EVALUATION

We now describe our experiments with e#. We first present the topics we extracted from a month of search queries. We then demonstrate e#'s effectiveness with a crowdsourcing study.

6.1 Topics of Expertise

We described how to extract topics of expertise with community detection. In this section, we illustrate our methodology with real world data. We use a full month of web search query logs (May 2014, US only, 998 GB). The graph we obtain contains approximately 60 million edges (1.45 GB). We describe our hardware setup in Section 6.3.

Figure 5 shows how many topics our algorithm finds after each iteration. We see that it starts with lots of tiny communities, then the count decreases very fast. Roughly, the procedure converges after 6 iterations. Figure 6 shows the distribution of the topic sizes. We observe that a large majority of communities contains between 2 and 10 queries (around 60%). We also found around 20% of orphans. We have very few communities with more than 50 items.

Figure 7 shows three groups of related keywords. To obtain this figure, we plotted the community which contains the term `49ers` (in dark blue), along with its three closest communities - in light blue, light green and dark green. The `49ers` community contains many non-trivial keywords: alternative spellings (`niners`), related activities

Set Name	Count	Examples
Sports	100	`49ers, hernandez, buffalo bills, nascar, baltimore ravens`
Electronics	100	`bluetooth, ipad mini, garmin, xbox, vacuum cleaners`
Finance	100	`nasdaq, dow futures, msft, quotes, bloomberg`
Health	100	`scoliosis, asthma, diabetes, bmi, bulimia`
Wikipedia	100	`world war II, aashiqui 2, lycos, beyonce, albert einstein`
Top 250	250	`sarah palin, mapquest, honda, antonov225, saudi arabia`

Table 1: Queries used for our crowdsourcing study.

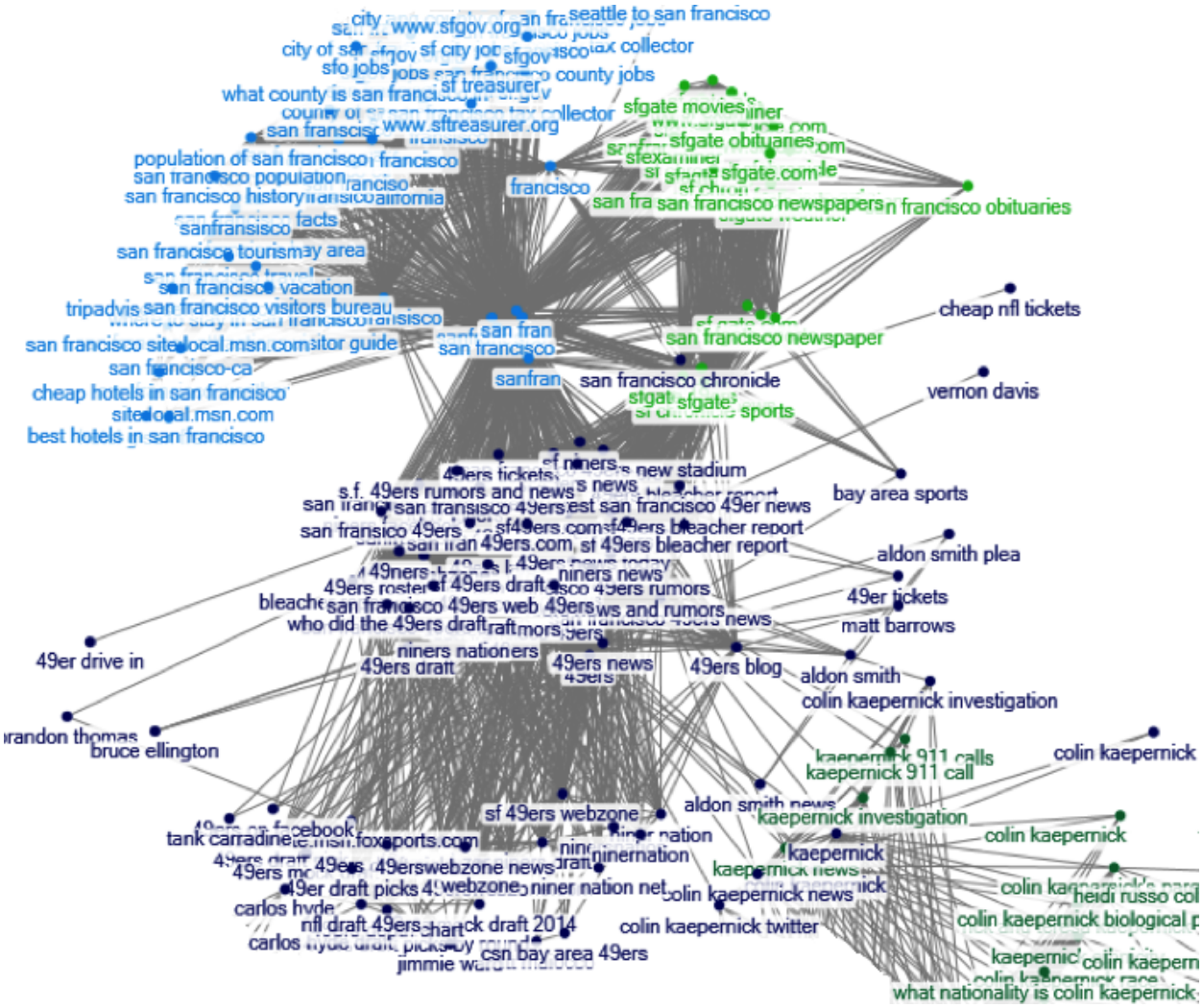


Figure 7: Graph and communities around the term "49ers".

(`49ers draft`), or players (`bruce ellington`, `vernon davis`). We see that the query-log distance lets us detect semantic associations - we could not have detected these relations with a string-based distance. The three other groups contain topics which are somehow related to the 49ers, but not closely enough to be used in query expansion. The light blue community contains topics related to San Francisco tourism. This is not surprising, because the 49ers is the official team of the city. The light green one mentions "SF Gate", which is a popular San Francisco newspaper (with a thick sports section). The dark green set focuses on Colin Kaepernick, a star player in the 49ers.

6.2 Impact on Expertise Retrieval

In this section, we demonstrate e#'s effectiveness on Web data with a crowdsourcing study.

6.2.1 *Experimental Setting*

We compare two algorithms: Pal and Counts' algorithm, detailed in the second section of this paper, and e#. To test the algorithms, we used queries which reflect popular interest in many different domains. Our assumption is that if a topic is popular on the Web in general, then it is likely to be popular on social media too. We used six sets, described in Table 1. The sets `Sports`, `Electronics`, `Finance`

Algorithm	Screen Name	Description	Verified	Followers
Baseline	SF49ersAllNews	All news about San Francisco 49ers	False	1,821
	Tim Kawakami	Tim Kawakami is a Mercury News sports columnist	True	45,924
	Matt Barrows	Matt (that's me) covers the 49ers for The Sacramento Bee.	True	36,271
e#	Tre9er	49ers/NFL/Draft Tweets. Host of NinersNation.com ..	False	4,651
	NinersGoldRush	Your source for all breaking 49ers news ...	False	4,135
	Red n Gold	Huge #49ers fan. LET'S GO #NINERS!	False	537

Table 2: Selected experts for the query `49ers`. The flag Verified comes from Twitter, it attests the authenticity of a popular account.

Algorithm	Screen Name	Description	Verified	Followers
Baseline	Huawei Club	Official Twitter handle for Huawei Club India...	False	1,589
	The Internet Patrol	The Internet Patrol is your source for Internet news.	False	179
	TekspeczDotnet	[...] where technology is not a passion, but an obsession.	False	87
e#	LuguLake	LuguLake believe tech shouldn't drift apart people ...	False	3,215
	HavitAworldnet	HAVIT is specialized in PC and entertainment ...	False	879
	Bluesound	High-res Music. Wireless. Everywhere.	False	1,308

Table 3: Selected experts for the query `bluetooth speakers`.

Algorithm	Screen Name	Description	Verified	Followers
Baseline	Arthur Hogan	Chief Market Strategist -Dad-SOX Fan	False	409
	CNBC Newsroom	... recognized world leader in business news	True	92,014
	WorldEconRecon	Breaking Financial News | Investment Analysis	False	7,470
e#	ET Commodities	Your most trusted resource for timely news...	True	15,733
	MarketWatch	Tracking the pulse of the markets...	True	1,119,485
	The Exchange	We promote financial literacy, through Hip Hop.	False	3,830

Table 4: Selected experts for the query `dow futures`.

Algorithm	Screen Name	Description	Verified	Followers
Baseline	Amer. Diabetes Assn.	Leading the fight to #StopDiabetes ...	True	73,905
	Diabetes 101	Get Educated About Type 1 Diabetes.	False	3,829
	DiabetesNews.com	The Most Comprehensive Diabetes News ...	False	47,402
e#	amidiabetic (Stuart)	Stuart #T1 diabetic, trying to help others	False	58,451
	Eliot LeBow	Psychotherapist & Certified Diabetes Educator	False	1,813
	Diabetesview	We deliver the latest Diabetes news everyday	False	6472

Table 5: Selected experts for the query `diabetes`.

Algorithm	Screen Name	Description	Verified	Followers
Baseline	National Interest	... premier international-affairs magazines.	False	12,340
	Franz-Stefan Gady	Foreign Policy Analyst, Occasional Reporter ...	False	1,054
	EmperorTigerstar	...YouTube channel about maps and history	False	116
e#	ProjectBugle	The First World War Commemoration Project	False	36
	Wales Remembers	1914-1918 Sharing stories, history, information	False	1,166
	WWI in Africa	What happened in Africa should not stay in Africa.	False	392

Table 6: Selected experts for the query `World War I`.

Algorithm	Screen Name	Description	Verified	Followers
Baseline	Ron Devito	Sarah Palin supporter; LAN Infrastructure PM	False	171
	Sarah Palin News	Palin news and opinion from Palin-focused sites.	False	19,897
	Jer	A Governor @SarahPalinUSA Conservative Supporter!	False	821
e#	Sarah Palin News	All news about Sarah #Palin	False	1,651
	TheDean'sReport	.. issues of the day from an honest [...] point of view	False	7,108
	Truthyism News	Truthyism is a news and media organization	False	177

Table 7: Selected experts for the query `Sarah Palin`.

Data set	Baseline	e#	Improvement
Sports	0.87	0.96	10%
Electronics	0.89	0.98	10%
Finance	0.94	0.97	3.1%
Health	0.82	0.98	19%
Wikipedia	0.83	0.87	4.8%
Top 250	0.64	0.86	35%

Table 8: Proportion of queries for which at least one candidate expert was found, before and after query expansion.

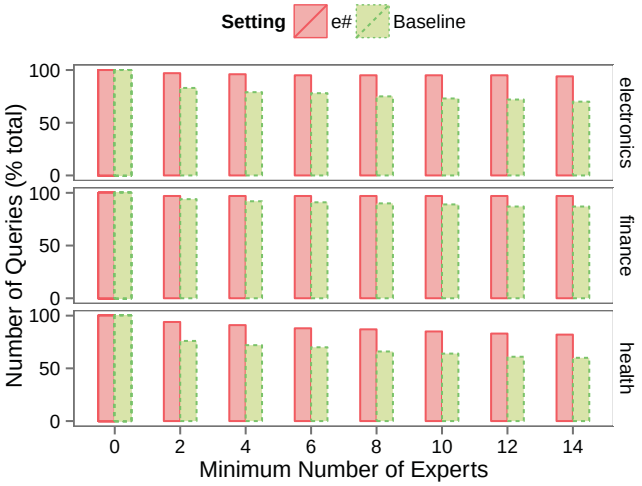
and Health contain the 100 most popular search terms from a commercial search engine, for each category. The set Wikipedia contains the title of the top 100 Wikipedia pages visited in 2014. It gives us an alternative view of popular interests. To increase diversity, we added the set Top 250, which contains the top 250 queries of a commercial search engine for July 2014. In total, we used 750 different queries.

We provide some examples of experts for the queries 49ers, bluetooth, dow futures, diabetes, World War I, and Sarah Palin in Tables 2, 3, 4, 5, 6 and 7 respectively. We observe that the experts are diverse: among others, we notice journalists (CNBC Newsroom), individuals (Arthur Hogan), support groups (Diabetes 101) and local associations (Wales Remember).

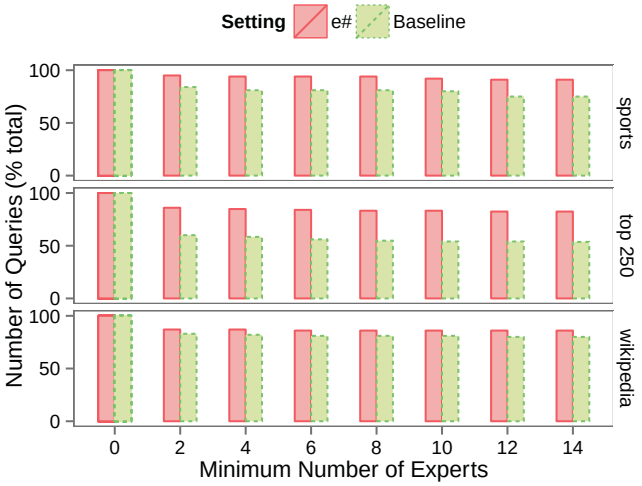
To assess the quality of the results, we were assisted by 64 crowdworkers, provided by a commercial third party. Evaluating expertise is a challenge for two reasons. First, the workers themselves must have some knowledge of the topic to recognize other experts. Second, the task is somehow subjective. We strived to incorporate these considerations in our experimental design. For each query, we generated up to 15 experts per algorithm and interleaved the results. To avoid worker fatigue, we chunked the resulting sets into smaller sets of at most 6 experts. We also randomized the order to prevent the position bias. We asked the workers to spot "non-experts", that is, accounts from which they could *not* get any objective information about the topic of interest. We chose to exclude "non-experts", rather than validate experts, because we assumed that the former task requires less background knowledge than the latter. We gave examples, encouraged high response times, presented links to the Twitter pages, and gave crowdworkers the option to ignore questions for which they were not confident. We filtered spammers with trivial preliminary questions. We set up the experiments such that each expert was reviewed by 3 different workers, and aggregated the results with majority voting.

6.2.2 *Impact on Recall*

In Table 8, we present the impact of query expansion on the size of the result sets. We show the number of queries for which at least one expert was found, before and after expansion. We note that in all six cases, we obtain a neat improvement. We notice the smallest performance gain with the Finance set: the baseline results are already very high, and e# only brings a 3% improvement. We observe the most dramatic effects with Top 250: e# answers 35% more queries. This result is not surprising: we trained e# on the search log from which the queries come from, therefore we expected it to perform well.



(a) Sets electronics, finance, and health.



(b) Sets commerce, top 250, and Wikipedia.

Figure 8: Effect of the query expansion on the number of experts per query.

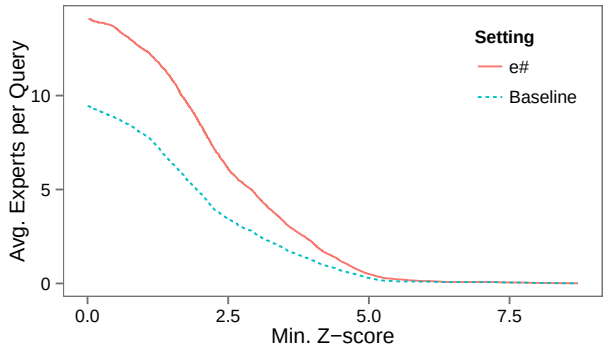
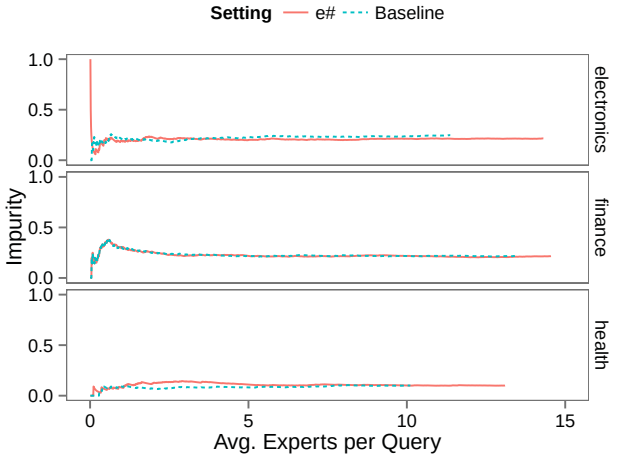
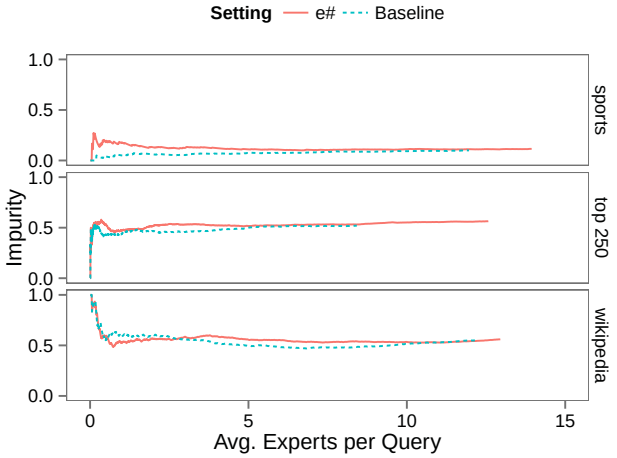


Figure 9: Impact of the z-score on the number of experts for the set Top 250.

Figure 8 present a finer view of e#'s impact on the number of experts retrieved for each query set. It presents the number of queries for which the algorithms return n experts



(a) Sets electronics, finance, and health.



(b) Sets commerce, top 250, and Wikipedia.

Figure 10: Size vs. quality trade-off. The impurity is the proportion of results marked as non relevant by the judges.

or more, with n varying between 0 and 14. For instance, the leftmost bars show that 100% of queries have 0 hits or more. The rightmost bars show the number of queries for which our algorithms found 14 experts or more. In almost every cases, we observe that query expansion improves the number of experts found (in average of about 10%, up to 30%). We conclude that our expansion strategy works: it does improve recall.

6.2.3 *Impact on Precision*

The major challenge in query expansion is to enhance recall without hurting precision. Indeed, query expansion may weaken the quality of the retrieved experts. This degradation has multiple sources: it comes from noise in the corpora, noise in the clustering, or errors in the expansion (e.g., disambiguation problems). In this section, we study the extent of this penalty.

Before we present our results, recall that our algorithm needs to be tuned. The users must choose a minimum z-score, under which the experts are rejected. This threshold defines a trade-off between precision and recall: a low value will lead to many low quality experts, a high value will lead to a few excellent experts. We illustrate this effect for `Top 250` in Figure 9.

Figure 10 compares the quality of the experts found for different levels of recall. For a given number of experts per query, it returns the *impurity*, that is, the proportion of experts marked as non relevant by the crowdworkers. We observe that the difference between the algorithms is very subtle. It is maximal for the first few results of the set `Sports`, it is almost imperceptible for `Finance` and `Health`. In the dataset `Electronics`, e# performs slightly better than its competitor. In conclusion, the accuracy penalty incurred by e# is minimal, if not negligible. We can improve expert detection recall with minimal losses in precision.

Step	VMs	Runtime	Read	Write
Extraction	65	38 min.	998 GB	2.6 GB
Clustering	65	2 hours	2.6 GB	94 MB
Expansion	1	$<$ 100 ms.	*	*
Detection	1	$<$ 1 sec.	*	*

Table 9: Resource consumption for one iteration (September 2015). As the number of virtual machines (VMs) allocated to each process varies in time, we report averages.

6.3 Resource Consumption

We now provides hints about the resource consumption of e#. The offline part of our system runs weekly on a production cluster. Table 9 reports statistics for one iteration (September 2015). It must be noted that our environment is completely virtualized, and thus performance depends heavily on availability: a relational operator can use between one and hundreds of virtual machines, depending on its nature and the cluster's workload. Besides, we have no control over the underlying hardware. The data center comprises servers with 12 x86-64 cores, between 32 and 64 GB main memory and SSD drives with 1 to 3 TB. But each of these servers can handle dozens of virtual machines. The collection of expertise topics produced by e# weighs about 100 MB. We store and index it in SQL Server 2014, which allows us to query it in a few milliseconds. We refer the reader to Pal and Counts [14] for more details about the final query detection.

7. RELATED WORK

Our work bridges two fields of study: expertise retrieval and query expansion. It is, to our knowledge, the first attempt to augment expert search with an external thesaurus.

7.1 Expertise Retrieval

Researchers have been interested in detecting experts for several years now, in particular since the problem was introduced at TREC in 2005 [2]. An extensive review was written by Balog et al. in 2012 [4]. The two oldest approaches are the *candidate model* and the *document model*. In the candidate model, a textual profile is created offline for each candidate (for instance, by aggregating all the documents authored by the candidate). Then, these profiles are ranked with traditional IR model [7]. The document model operates the other way around. First, a set of relevant documents is identified for the query. Then, the algorithms find the associated people and rank them according to the relevance of the documents and their degree of asso-

ciation [15]. More recently, authors have proposed alternative models. Discriminative models such as the Arithmetic Mean Discriminative model are robust and they can integrate heterogeneous, arbitrary features [9]. Graphs models have also gained popularity. For instance, Serdyukov et al. have shown the effectiveness of random walks on "expertise graphs" [18].

Jianshu et al. is, to our knowledge, the first team to have published work about expert detection on Twitter [20]. Their system is based on a graph describing the topical similarity between the users. To detect authorities, they run a variant of PageRank on this graph for each topic. An alternative was proposed by Pal et al. [14]. We introduce a production version of their framework. Recent work has studied how to incorporate location data into expertise retrieval, focusing on "local" experts rather than "general" experts [6].

As our framework is based on query expansion, we do not compete with any of these approaches. Our system can work with any Expertise Retrieval system.

7.2 Query Expansion

Authors have proposed query expansion methods for decades in document search. Researchers were already building "classes of similar terms" to improve search before 1960 [16]. To measure the proximity between terms, they used co-occurence in the training documents. Qiu et al. proposed a notable improvement with "concept-based" query expansion [16]. The main idea was to represent the terms by points in a vector space, where each dimension represents a document. From this representation, it was possible to build a so-called similarity thesaurus. More recent publications have shown that external source of knowledge can also improve search, such as WordNet [19] or ontologies [12].

Our work differs from all of the above because we use a query log. This source of data is relevant for two reasons. First, it is relatively easy to manipulate: we do not have to process the whole collection of documents (in our case, this would mean the whole Web). Second, it is constantly renewed; thus, we believe that the microblogging vocabulary is better captured by this source than by existing ontologies. Other authors have used query logs for query expansion, such as Cui et al. [8]. They observe that if a set of documents is frequently selected for a certain keyword, then their terms are probably strongly correlated to this keyword. However, they still use the underlying documents.

It seems that little work was presented on query expansion in the context of expertise retrieval. Macdonald et al. have mostly focused on local query expansion, i.e., using top ranked documents for pseudo-relevance feedback [11]. Balog et al. have presented ways to incorporate external evidence of expertise into language models [3]. These lines of work are complementary to ours, but they are not overlapping.

8. CONCLUSION

We introduced an approach for expertise detection on social media that emphasizes recall. We showed that finding related domains of interests can be expressed as a graph community detection problem. We presented a parallelized implementation and showed the evaluation results on a large Twitter data set. Our findings demonstrated that e# can increase the number of experts without losing quality.

Future work includes expanding into other social networks such as Quora and Facebook, and exploring different community detection paradigms.

9. REFERENCES

[1] R. Baeza-Yates and A. Tiberi. Extracting semantic relations from query logs. *Proc. SIGKDD*, page 76, 2007.

[2] P. Bailey, A. D. Vries, N. Craswell, and I. Soboroff. Overview of the TREC 2007 Enterprise Track. *TREC*.

[3] K. Balog and M. de Rijke. Non-local evidence for expert finding. In *Proc. CIKM*, pages 489–498, 2008.

[4] K. Balog, Y. Fang, M. de Rijke, P. Serdyukov, and L. Si. Expertise Retrieval. *Foundations and Trends in Information Retrieval*, pages 127–256, 2012.

[5] R. Chaiken, B. Jenkins, P.-Å. Larson, B. Ramsey, D. Shakib, S. Weaver, and J. Zhou. SCOPE: easy and efficient parallel processing of massive data sets. *Proc. VLDB*, pages 1265–1276, 2008.

[6] Z. Cheng, J. Caverlee, H. Barthwal, and V. Bachani. Who is the barbecue king of texas?: a geo-spatial approach to finding local experts on twitter. In *Proc. SIGIR*, pages 335–344, 2014.

[7] N. Craswell, D. Hawking, A.-M. Vercoustre, and P. Wilkins. Panoptic expert: Searching for experts not just for documents. In *Ausweb Poster Proc.*, 2001.

[8] H. Cui, J.-R. Wen, J.-Y. Nie, and W.-Y. Ma. Probabilistic query expansion using query logs. In *Proc. WWW*, pages 325–332, 2002.

[9] Y. Fang, L. Si, and A. P. Mathur. Discriminative models of integrating document evidence and document-candidate associations for expert search. In *Proc. SIGIR*, pages 683–690, 2010.

[10] S. Fortunato. Community detection in graphs. *Physics Reports*, pages 75–174, 2010.

[11] C. Macdonald and I. Ounis. Expertise drift and query expansion in expert search. In *Proc. CIKM*, pages 341–350, 2007.

[12] R. Navigli and P. Velardi. An analysis of ontology-based query expansion strategies. In *Proc. ECML, Workshop on Adaptive Text Extraction and Mining*, pages 42–49, 2003.

[13] M. E. J. Newman. Modularity and community structure in networks. *Proc. National Academy of Sciences*, 2006.

[14] A. Pal and S. Counts. Identifying topical authorities in microblogs. *Proc. WSDM '11*, page 45, 2011.

[15] D. Petkova and W. B. Croft. Hierarchical language models for expert finding in enterprise corpora. *IJAIT*, pages 5–18, 2008.

[16] Y. Qiu and H.-P. Frei. Concept based query expansion. In *Proc. SIGIR*, pages 160–169, 1993.

[17] E. J. Riedy, H. Meyerhenke, D. Ediger, and D. A. Bader. Parallel community detection for massive graphs. In *Parallel Processing and Applied Mathematics*, pages 286–296, 2012.

[18] P. Serdyukov, H. Rode, and D. Hiemstra. Modeling multi-step relevance propagation for expert finding. In *Proc. CIKM*, pages 1133–1142, 2008.

[19] E. Voorhees. Query Expansion using Lexical-semantic Relations. In *Proc. SIGIR*, pages 61–69, 1994.

[20] J. Weng, E.-P. Lim, J. Jiang, and Q. He. Twitterrank: finding topic-sensitive influential twitterers. In *Proc. WSDM*, pages 261–270, 2010.