Conkar: Constraint Keyword-based Association Discovery

Mo Zhou
Indiana University, USA
mozhou@cs.indiana.edu

Yifan Pan
Indiana University, USA
panyif@cs.indiana.edu

Yuqing Wu
Indiana University, USA
yuqwu@cs.indiana.edu

ABSTRACT

In many domains, such as bioinformatics, cheminformatics, health informatics and social networks, data can be represented naturally as labeled graphs. To address the increasing needs in discovering interesting associations between entities in such data graphs, especially under complicated keyword-based and structural constraints, we introduce Conkar (**Con**strained **K**eyword-based **A**ssociation Discove**R**y) System. Conkar is the first system for discovering *constrained acyclic paths (CAP)* in graph data under keyword-based constraints, with the highlight being the set of quantitative constraint metrics that we proposed, including *coverage* and *relevance*. We will demonstrate the key features of Conkar: powerful and user-friendly query specification, efficient query evaluation, flexible and on-demand result ranking, visual result display, as well as an insight tour on our novel CAP query evaluation algorithms.

Categories and Subject Descriptors

H.3.3 [**Information Systems**]: Information Storage and Retrieval—*Information Search and Retrieval (Search process)*

General Terms

Algorithms

Keywords

Keyword-based Association Discovery, Search Algorithm, Coverage, Relevance, Drug Discovery

1. INTRODUCTION

In many domains, such as social networks, cheminformatics, bioinformatics, and health informatics, data can be represented naturally in graph model, with nodes being data entities and edges the relationships between them. RDF (Resource Description Framework) [11] is a W3C recommended language which describes linked data of the Semantic Web in the form of triples. Both RDF data and RDF schema can be represented by node and edge labeled graphs.

The graph nature of these data brings opportunities and challenges to data storage and retrieval. In particular, it opens the doors to search problems such as semantic association discovery [3, 4] and keyword search [7].

We study the application requirements in various domains including cheminformatics and social networks and find that the problem of discovering acyclic paths between data entities under constraints such as appearance of nodes, edges and/or patterns and the length of paths is at the core of these applications.

One of the core research directions in cheminformatics is to study drugs' affect on diseases via the interaction between proteins and genes [10]. To facilitate such study, drugs, diseases, proteins, genes, etc. are represented in an RDF graph [5]. To assess the effectiveness of a drug before conducting chemical experiments, domain experts would like to first find and analyse paths from the drug to the target disease under constraints, e.g. the (partial) appearance of certain proteins/genes.

In social networks and research networks, such as VIVO [6], one of the most important functionalities is to find connections between people that satisfy certain constraints [1, 8], for example, to find possible collaborations based on common interest, or to detect conflict of interest between authors and PCs based on the appearance of certain entities, such as professional institutions that both are affiliated with, and/or relationships, e.g. advise, co-author and co-PI. It is also critical to determine the degree of such conflict of interest further based on how frequent certain relationship and entities appear in the connection [1].

How to express and measure constraints and how to answer such constrained path discovery queries efficiently are critical problems, but are not yet fully studied in the literature. There are no generic systems that fulfill the demands. First, allow us to take social/research networks as example, to illustrate such demands.

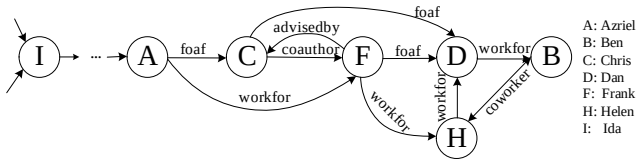

Figure 1: Graph representation of a sample RDF data

EXAMPLE 1.1. *Figure 1 shows the graph representation of a sample RDF data representing the relationships among people in the social networks. Let's consider the following search requests:*
case1. *Find how* Azriel *connects to* Ben*;*
case2. *Find the close ties (within 3 steps) between* Azriel *and* Ben*;*
case3. *Find how* Azriel *connects to* Ben *through* Chris *or* Dan*;*
case4. *Find how* Azriel *connects to* Ben *through at least two people from* Chris, Dan *and* Ida *within four steps.*
case5. *Find* Azriel*'s close (within 4 steps) professional connections (e.g. relationships such as* workfor, coworker, coauthor*) to* Ben*;*
case6. *Find* Azriel*'s close (within 4 steps) semi-professional con-*

CIKM'11, October 24–28, 2011, Glasgow, Scotland, UK.

nections to Ben *(i.e. half of the relationships in any tie should be professional);*

Query languages have been proposed to query data on the Semantic Web. SPARQL [12], the de facto standard query language for RDF, relies on graph patterns to identify data entities and relationships of interest. However, it lacks the ability to express arbitrary paths between data entities, such as those shown in Ex. 1.1. The notion of label-constraint reachability (LCR) [8] was proposed to describe the problem of finding the *connectivity* between two given nodes in a labeled graph under the constraint that all the edge labels along the path are in a given set. The semantic keyword search problem was defined to find trees in a labeled directed graph where the tree nodes and edges cover all the keywords [7]. Combining these notions, several SPARQL extensions, with the introduction of path variables, were proposed [4, 9]. Such extensions are capable of expressing the search queries in cases 1-2 in Ex. 1.1, but not the other cases. There were also proposals for extending SPARQL with regular expressions [2] to express complex patterns that satisfy strictly defined constraints such as cases 1,2,3,5. However, such SPARQL extensions still cannot express more relaxed constraints such as those in cases 4,6.

To support the searches in Ex. 1.1, the two major challenges are: how to express these association discovery problems and how to answer them efficiently.

We introduce Conkar [14], the first system that can express and answer all these search queries[1]. In Conkar, whose architecture is shown in Figure 2, we abstract the association discovery problem as the problem of *finding constrained acyclic paths (CAP) in directed labeled graphs*. We define a set of quantitative metrics, at the core of which are the notions of *coverage* and *relevance* of a path with respect to a set of keywords. Via the easy-to-use interface of Conkar, users can express CAP search queries by specifying keyword sets, the desired *coverage* and *relevance* of the resultant paths with respect to these keyword sets, as well as other constraints, such as path length. Users can also specify the relative importance of quantitative metrics. Then Conkar will rank the results based on their preference. Upon viewing the results, users can adjust (extend or refine) the constraints and ranking criteria to obtain the results that suit their needs.

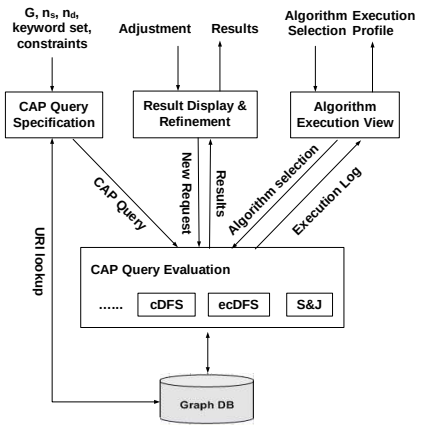

Figure 2: Conkar architecture

To address the challenges in answering the CAP queries efficiently on large graph data, we introduce a family of CAP discovery algorithms, including DFS-base algorithms constraint-DFS (cDFS) and enhanced cDFS (ecDFS), and a localized search-and-join (S&J) algorithm. All of them are designed to minimize the search space by taking advantage of the constraints to eliminate unpromising search branches as early as possible.

2. EXPRESSING CAP SEARCH QUERIES

[1]Conkar overview: http://www.cs.indiana.edu/~mozhou/conkar.html

The important innovations of Conkar are to formally define the *Constrained Acyclic Path Discovery* problem and to provide a friendly interface for users to express CAP queries.

2.1 Preliminary

Let $\mathcal{L}$ be an infinite set of literals and $\mathcal{U}$ be an infinite set of URIs disjoint with $\mathcal{L}$. We represent RDF data as a node and edge labeled directed graph $G = (V, E, \lambda)$ where V is a set of nodes, $E \subset V \times V$ is a set of edges, and λ is a labeling function that maps items in $V \cup E$ into a finite set of labels and literals.

We call a sequence of interleaving nodes and edges of graph G a *path fragment* (or *fragment*), represented by f, if

- for every adjacent (n, e, n') in f, it is the case that $n, n' \in V \wedge e = (n, n') \in E$;
- if (e, n) is a prefix of f, then there must exist $n' \in V$, such that $e = (n', n) \in E$; and
- if (n, e) is a suffix of f, then there must exist $n' \in V$, such that $e = (n, n') \in E$.

We use $nodes(f)$ ($edges(f)$) to represent the set of nodes (edges) in f, and $Length(f)$ (or $|f|$) the length of f, defined as $|edges(f)|$. We overload the mapping function λ to map a set of nodes (edges) to their corresponding labels.

Given two nodes $n_s, n_d \in V$ as the source and destination nodes, the paths that link n_s to n_d in G are fragments in the form $f = (n_s, e_1, n_1 \ldots, n_{k-1}, e_k, n_d)$. In search queries that look for the paths between two nodes, frequently only acyclic paths are of interest to users. Therefore, as n_s and n_d are given, we are particularly interested in a special type of path fragment - acyclic e-fragment (denoted by f_e), which starts and ends with an edge, and no two nodes, including the end nodes, are the same. We use $\mathcal{F}_e(n_s, n_d)$ to represent all acyclic e-fragments between n_s and n_d in G.

2.2 Specifying Constraints

Length Constraint: Among a possibly large number of resultant e-fragments of a search query, shorter e-fragments tend to express stronger and more meaningful relationship between the two end nodes than the longer ones do [3, 7]. Involving the *length constraint* which restricts the length of the resultant e-fragments has been studied in [2, 4], and Conkar supports it as well.

Set-based Constraints: When users search for paths between a pair of nodes, they are frequently interested in paths that contain certain keywords. Such constraints can be given in the form of a *keyword set* (denoted $\mathcal{S}$), where a keyword is a label in $\mathcal{U}$.

Certain constraints, such as presence constraints [4] on nodes and tight constraints on edges [8] were discussed in the existing works, but they are quite limited in terms of what can be in the keyword set and how the results are regulated by it, hence can express only some search queries, but not many others, such as cases 3-6 in Ex. 1.1.

We generalize the keyword-based constraints defined in [4, 8] in two directions: (1) we allow the keyword set to include keywords that can be mapped to labels of both nodes and edges; and (2) we extend how the results are confined by a keyword set.

DEFINITION 2.1. *Given a finite keyword set $\mathcal{S} \subseteq \mathcal{U}$ and an e-fragment $f_e \in \mathcal{F}_e(n_s, n_d)$,*

1. *if $\mathcal{S} \subseteq (\lambda(nodes(f_e)) \cup \lambda(edges(f_e)))$, we say f_e satisfies* presence constraint *w.r.t. $\mathcal{S}$;*
2. *if $\mathcal{S} \supseteq (\lambda(nodes(f_e)) \cup \lambda(edges(f_e)))$, we say f_e satisfies* context constraint *w.r.t. $\mathcal{S}$;*
3. *if $\mathcal{S} \cap (\lambda(nodes(f_e)) \cup \lambda(edges(f_e))) \neq \emptyset$, we say f_e satisfies* intersection constraint *w.r.t. $\mathcal{S}$.*

Based on the definition above, we can express the search request in Ex. 1.1 case3 as "find e-fragments from *Azriel* to *Ben* that satisfy the *intersection constraint* w.r.t. keyword set $\{Chris, Dan\}$".

Quantitative Metrics: To express the search requests in Ex. 1.1 cases 4-6, a more subtle description of the relationship between an e-fragment and a keyword set than the *all-or-nothing* set-based constraints described in Def. 2.1 is needed. For this purpose, we introduce quantitative metrics *coverage* and *relevance*.

Intuitively, *coverage* describes the fraction of the keyword set that appears in the label set of an e-fragment, while *relevance* describes the fraction of the labels of an e-fragment that are in the keyword set. In an RDF graph, each node has its *unique* label, while more than one edge may have the same label. Therefore, we refine *coverage* and *relevance* further into *node-coverage* and *node-relevance* for keyword sets to be applied only on nodes, *edge-coverage* and *edge-relevance* for edges, and use *coverage* and *relevance* for the constraints in which keywords can be mapped to both nodes and edges. We use $cntE(l, f_e)$ to represent the number of appearance of a keyword l among the edges in an e-fragment f_e.

We present the formula to calculate the coverage and relevance in Def. 2.2 while the formula to calculate node (edge) coverage and relevance can be easily deduced.

DEFINITION 2.2. *Given a graph G, two nodes $n_s, n_d \in V$ and a finite keyword set $\mathcal{S} \subseteq \mathcal{U}$, for an e-fragment $f_e \in \mathcal{F}_e(n_s, n_d)$,*

$$Coverage(f_e, \mathcal{S}) = \frac{|\mathcal{S} \cap (\lambda(nodes(f_e)) \cup \lambda(edges(f_e)))|}{|\mathcal{S}|} \quad (1)$$

$$Relevance(f_e, \mathcal{S}) = \frac{|\mathcal{S} \cap \lambda(nodes(f_e))| + \Sigma_{l \in \mathcal{S}} cntE(l, f_e)}{|nodes(f_e)| + |edges(f_e)|} \quad (2)$$

Taking advantage of the quantitative metrics, the search requests in Ex. 1.1 cases 4-6 can be expressed easily. For example, case 6 can be expressed as "find all e-fragments, f_e, from *Azriel* to *Ben* such that EdgeRelevance(f_e, {*coworker*, *workfor*, *coauthor*}) $\geq$ 0.5 and $|f_e| \leq 4$".

All the constraints we defined in Def. 2.1 are special cases that can be expressed using the quantitative metrics defined in Def. 2.2.

$$Presence(f_e, \mathcal{S}) \iff Coverage(f_e, \mathcal{S}) = 1 \quad (3)$$

$$Context(f_e, \mathcal{S}) \iff Relevance(f_e, \mathcal{S}) = 1 \quad (4)$$

$$Intersection(f_e, \mathcal{S}) \iff Relevance(f_e, \mathcal{S}) > 0 \quad (5)$$

Similarly, we can define the node/edge version of these functions.

2.3 CAP Query Specification

Formally, a CAP search query $CAP(n_s, n_d, \tau)$ on RDF graph G involves source and destination nodes $n_s, n_d \in V$, and constraint τ expressed using zero to many quantitative metrics functions that involve zero or many keyword sets, and returns the e-fragment(s) from n_s to n_d that satisfy τ. Our Conkar provides a user-friendly interface for specifying a CAP query by composing the ingredients of the search, i.e. G, n_s, n_d and an optional, potentially arbitrarily complicated τ.

3. ANSWERING CAP QUERIES

As the data graph G and constraint τ can both be large in size and complex in nature, designing algorithms that can efficiently answer CAP queries is at the heart of the design and implementation of Conkar.

Many graph searching algorithms [7] were proposed for answering keyword searches in graph data. But they can not be applied directly to answer CAP queries, as they are designed to find tree instances covering all keywords in the graph. Traversal-based approaches such as DFS, BFS and bidirectional search [2] can be adapted with a post-search filter to answer CAP queries. However, such Search-Filter approaches don't scale well when the size and complexity of the graph increase. Algorithms [3, 4] with additional help from schema or indices were later proposed, but these facilitating data and structures lack of either availability or scalability.

In Conkar, we implemented three algorithms to answer CAP queries:

Search-Filter approach utilizes algorithms proposed in the literature to find all acyclic e-fragments in $\mathcal{F}_e(n_s, n_d)$ (in fact, to find $CAP(n_s, n_d, \emptyset)(G)$), and then eliminates those not satisfying the constraints specified in τ. This approach is most generic and can be easily adapted for all constraints introduced above. However it is not practically efficient because generating $\mathcal{F}_e(n_s, n_d)$ is very time and space consuming, rendering the *search* phase costly, while frequently $|CAP(n_s, n_d, \tau)(G)| \ll |CAP(n_s, n_d, \emptyset)(G)|$, rendering the high cost of the *search* phase mostly wasted.

In Conkar, to answer CAP queries efficiently, we propose to take advantage of the constraints on length and keywords and push result validation into the path discovery process, to prune unpromising intermediate results as early as possible. In particular, we design and implement the following two families of algorithms [13].

ConstraintDFS (cDFS) and *Enhanced ConstraintDFS* (ecDFS) algorithms are based on *Depth First Search* (DFS). To minimize the DFS search space in computing a CAP query $CAP(n_s, n_d, \tau)$, we minimize the total number of intermediate results, i.e. fragments, generated in the process by terminating expansion from fragments generated at search steps when we are certain that the expansion will not lead to any results. In order to do so, we introduce the notion of *projected value range* of the quantative metrics, and we design a set of formula to compute the projected value ranges of the constraint metrics in the CAP query in question, based on information, such as the length and keyword appearances, of the fragments generated so far in the DFS process. In the cDFS algorithm, at each DFS step, when the projected value ranges of a fragment do not overlap with the constraints in τ, we can safely terminate the expansion of this fragment. The ecDFS algorithm further improves the performance by saving the validation of the fragments during the search steps of which we foresee the fragments guaranteed to be promising.

Search-and-Join (S&J) algorithm specifically targets at the set-based, rather than list-based, keyword constraint specification. The S&J algorithm takes the local information around the nodes with keywords into consideration to calculate more precise projected value ranges, and thus conduct more efficient pruning. The S&J algorithm issues mini-searches to find exclusive path fragments (i.e. paths that do not pass through any keyword nodes) between pairs of nodes that contain keywords, then use *constrained sequence join* to concatenate the fragments to produce the final results. Careful bookkeeping allows us to use the partial results in one mini-search to limit the search space of many other mini-searches, and effectively reduces the overall search cost.

4. DEMONSTRATION PROPOSAL

We propose the demonstrational plan to welcome users to have a hand-on experience with Conkar. Especially users can (1) issue CAP queries, simple or complicated, via our user-friendly interface and view the resultant paths instantly; and (2) have an insight look at how our algorithms efficiently evaluate CAP queries. We will demonstrate Conkar on two applications: the drug discovery application for chemical informatics and the application on social/re-

search networks. Following the running examples in Sec. 1, we will again use social/research networks data as our example here.

4.1 CAP Query Specification and Answering

Users are welcome to use the CAP query interface of Conkar to specify and evaluate CAP queries. The required fields are (1) the data graph, which can be selected from our demo databases; and (2) the source and destination nodes, which can be specified by typing keyword(s) in a textbox, and selecting URIs that Conkar pulls from the database matching the keyword(s). The constraints are optional. Using the simple CAP query interface of Conkar, users can construct a set of keywords, and use scroll bars to set the desired values ranges of the quantitative metrics with respect to the keyword set. Users can also specify the importance of each constraint, which Conkar will take into consideration while presenting the ranked results to reflect users' preferences. Conkar visualizes the ranked results in list view as well as graph view. The simple search interface and result display are shown in Figure 3.

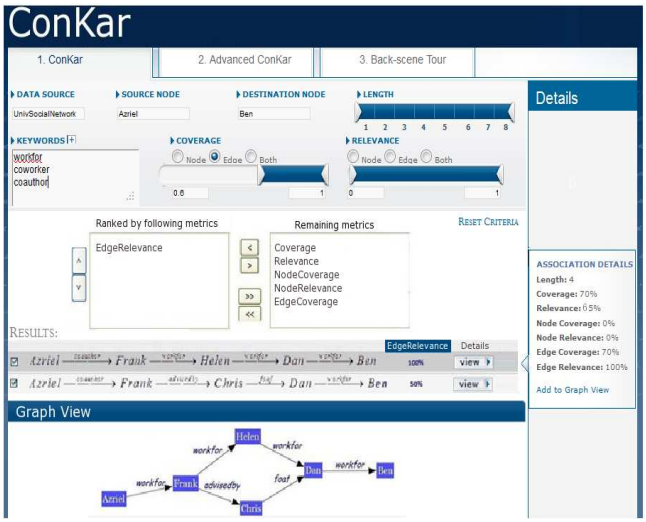

Figure 3: Simple search

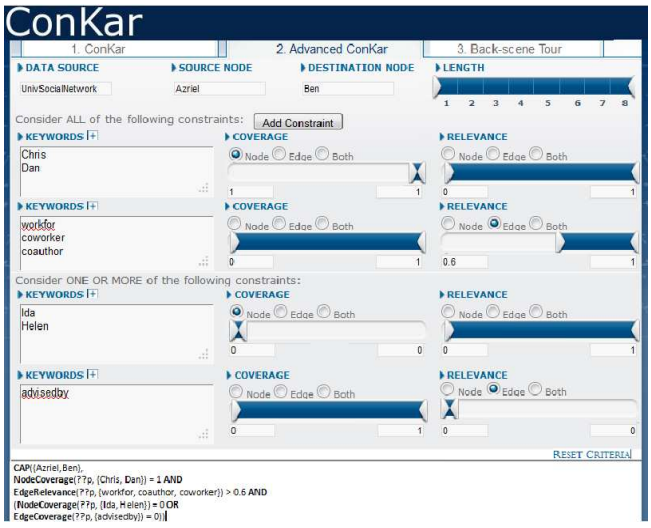

Figure 4: Advanced search

More sophisticated users are welcome to use the advanced CAP query interface to specify complex CAP search queries, consisting of constraints based on different keyword sets, linked via logical operators (AND and OR). Conkar also offers the resultant CAP query generated from the click-and-drag interface in the CAP query box for users to inspect. Figure 4 shows a CAP query under construction using the advanced search of Conkar.

4.2 Query Evaluation Behind-the-Scene

Besides demonstrating the key features of Conkar, e.g. CAP query specification and execution, we also provide a behind-the-scene tour for audiences who are interested in our CAP query evaluation algorithms. Here, after composing a CAP query, we will welcome the audience to choose an algorithm to evaluate the query. Conkar will then illustrate how nodes/edges in the data graph are explored, the intermediate results, as well as our sophisticated bookkeeping that facilitates the search. At the end, besides showing the query results, Conkar will also report the execution profile, including how many node/edges are explored, using which the users can compare the performance of different algorithms. Figure 5 illustrates an intermediate step of using the S&J algorithm to answer a CAP query. Here, nodes are color coded to distinguish source/destination nodes, query nodes, search frontiers, pruned branches, etc.

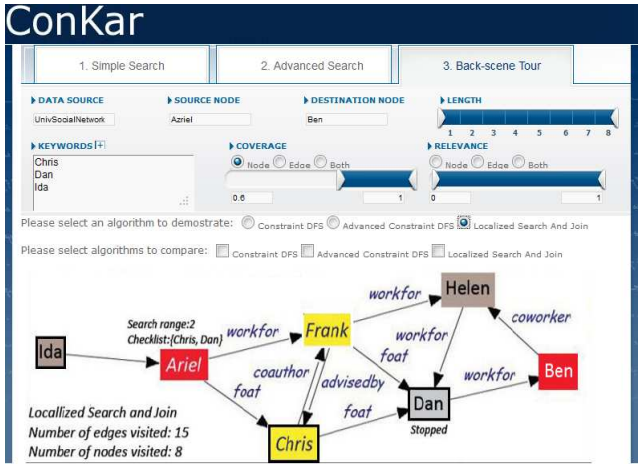

Figure 5: Behind the scene

5. REFERENCES

[1] B. Aleman-Meza, *et al.* Semantic Analytics on Social Networks: Experiences in Addressing the Problem of Conflict of Interest Detection. In *WWW*, 2006.

[2] F. Alkhateeb, *et al.* Constrained Regular Expressions in SPARQL. In *SWWS*, 2008.

[3] K. Anyanwu, *et al.* ρ-Queries: Enabling Querying for Semantic Associations on the Semantic Web. In *WWW*, 2003.

[4] K. Anyanwu, *et al.* SPARQ2L: Towards Support for Subgraph Extraction Queries in RDF Databases. In *WWW*, 2007.

[5] C. Bin, *et al.* Chem2Bio2RDF: a Semantic Framework for Linking and Data Mining Chemogenomic and Systems Chemical Biology Data. In *BMC Bioinformatics*, 2010.

[6] Y. Ding, *et al.* Semantic Web Portal: A Platform for Better Browsing and Visualizing Semantic Data. In *AMT*, 2010.

[7] H. He, *et al.* BLINKS: Ranked Keyword Searches on Graphs. In *SIGMOD*, 2007.

[8] R. Jin, *et al.* Computing Label Constraint Reachability in Graph Databases. In *SIGMOD*, 2010.

[9] K. Kochut, *et al.* SPARQLeR: Extended SPARQL for Semantic Association Discovery. In *ESWC*, 2007.

[10] S. Lee, *et al.* Building the Process-drug-side Effect Network to Discover the Relationship Between Biological Processes and Side Effects. In *BMC Bioinformatics*, 2011.

[11] F. Manola, *et al.* RDF Primer. *W3C Recommendation*, 2004.

[12] E. Prud'hommeaux, *et al.* SPARQL Query Language for RDF. *W3C Recommendation*, 2008.

[13] M. Zhou, *et al.* Efficient Association Discovery with Keyword-based Constraints on Large Graph Data. In CIKM, 2011.

[14] Conkar: http://www.cs.indiana.edu/~mozhou/conkar.html