A Model for Automatic Generation of Multi-partite Graphs from Arbitrary Data

Ricardo Baeza-Yates
Yahoo! Research
Barcelona, Spain

Nieves Brisaboa
Univ. of A Coruña
A Coruña, Spain

Josep Larriba-Pey
Universitat Politècnica de Catalunya
Barcelona, Spain

March 31, 2010

Abstract

In this paper we propose a generic model to generate basic multi-partite graphs obtained by associations found in arbitrary data. The interest of such a model is to be the formal basis behind a tool for automatic graph generation that we are developing. This tool will automatically generate the basic multi-partite graphs that represents the arbitrary data provided as input.

We consider input data as collections of complex objects composed by a set or a list of heterogeneous elements. Our tool will provide an interface for the user to specify the kind of nodes that are relevant for the application domain in each case. Those nodes will be obtained from the complex input objects by simple *extraction* rules.

The objective of this paper is only to present the model to represent basic multi-partite graphs and the way to define the nodes of the graph using simple *derivation rules* defined by the user.

In order to validate our model we give three examples of radically different data sets. Those examples come from the Web log queries, processing text collections, and bibliographic databases.

1 Introduction

The use of graphs for data analysis becomes essential in many situations. Plain texts, for instance, formed by paragraphs, sentences and words at the simplest, have no structure but giving them a graph structure may lead to better analysis of their characteristics [11]. In other situations, like bibliographic analysis [10], where data are structured, or in web structure analysis [7], the use of graphs gives a significant insight that allows for deeper and faster analysis.

Also, the data to be analyzed grows in size and in the number of varied sources, making graphs and the use of graph analysis commonplace in some situations. Thus, at times, the use of graphs may provide the more easy and understandable artifact to provide simple views of the data and their relationships. In other cases, however, the complexity of the graphs obtained from plain or structured data may deter the proper analysis, or even the capability of computers to process them. Thus, it seems obvious that creating graphs in a simple way will be necessary since graphs are becoming important in many situations.

However, creating a graph from a data collection is not a trivial nor a fast operation. Therefore, tools for the automatic generation of basic graphs from a general collection of input data will be

very useful and welcome in the future. In order to do so, it is first necessary to create the ground basis for these tools.

The objective and main contribution of this paper is to set the basis for the creation of tools for the automatic generation of graphs from basic datasets in the framework of a broader research project oriented to high performance graph management and analysis [3].

The type of tool we are planning will allow the user to define the kind of nodes and relationships of interest. Thus, a single data set, by means of using different transformations and rules, will lead to different graphs that are, in a sense, views of a more generic graph that suit and simplify further analytic queries on the resulting product graphs.

The rest of the paper is organized as follows. Section 2 covers related work. The model is presented in Section 3, while in Section 4 we present three examples of its use. We end in Section 5 with some final remarks.

2 Related work

Graphs are a general mathematical model that formalizes each connection between two entities and emerges anywhere where we encode or represent that such two entities are associated. But the modelling can occur between different entity sets, for example which user visits which web site. Thus, graphs can model bibliographical databases, XML databases, legal databases or the Web itself. In fact, link analysis can be considered the birth of graph mining in the Web, with PageRank and HITS being the main simple landmark results. In the context of the Web, graphs are dynamic, changing over time and also with different characteristics in different cultural and geographical scopes. In social networks, graph mining also enables the study of populations' behaviour and finding communities. Successful graph mining not only enables segmentation of users but also enables prediction of behaviour. Therefore, graph mining remains an area of high interest and high development. A detailed picture with respect to algorithms and potential for graph mining is presented by Chakrabarti and Faloutsos [8]. The structure of their survey focuses around what are the graph patterns that seem to be frequent in real graphs reflecting the many domains of human behavior model in them. Interestingly, the survey points out that algorithms for computing community effects, cluster coefficients and the like have emerged from many disciplines and this is the aspect of graph mining that reflects the largest impact as well as the potential from cross-discipline fertilization. Another important work is the graph mining book by Cook and Holder [9].

Current research has mainly focused on efficiency aspects to handle the scalability issues that arise in very large graphs, for example, stream processing models that can mine a graph in linear time [15]. Here, we look at the modeling side, so we can define and prototype applications to test hypothesis or to do exploratory research before handling large volumes of data. This is similar to the spirit of WIM, where Web mining applications can be prototyped very fast using a high-level programming language [14].

3 A Model for Basic Multi-Partite Graphs

The generic model we propose in this paper was designed to support the general tool we are developping. This tool will automatically create the basic multi-partite graphs that can be obtained applying *derivation rules* over collections of objects presented as input. Each kind of *input* object can be a set or a list of heterogeneous elements. That is, our goal is to implement a tool to

automatically generate an initial and basic multi-partite graph from the collections of complex objects(set or list of elements) provided as inputs. The generated graphs, that the tool we are developing will represent using GraphML [6], could be used later for more complex manipulations and queries using general graph management tools such as Neo4j [13] or DEX [12].

Our model distinguishes three levels of objects that will represent the nodes of the graph. We call those: *Input* objects (I), *Derived* objects (D) and *Atomic* objects (A) depending on the level they occupy. All the objects, except those provided as input, are generated using *derivation or extraction rules*R. Those rules will be defined by the user according to the characteristics and needs of the application domain. The rules are always applied over objects of some type and produce objects of other types with, usually, a lower granularity level. The tool will apply such rules R_i over the collection(s) of input objects I_i, obtaining derived objects D_j of the different types defined by the user. In the same way, the atomic elements A_k belonging to the different defined atomic types will be obtained applying derivation rules over derived or input objects. Normally, the input data will be at one extreme of the multi-partite graph while the atomic elements will be at the other extreme, representing the highest and lowest levels possible respectively.

Summarizing, the main elements of our model are:

- **Derivation rules R_i.** Those are user defined procedures that the tool will use to produce derived and atomic objects from the input objects. The tool will provide simple algorithms for the user to define rules such as "extract all the words from a document" or "eliminate stopwords". The interesting derivation rules are domain dependant, therefore, the tool will allow for the definition of more complex derivation rules that will use external procedures provided by the user. Notice that the relationships among an object and those objects derived from it are defined by the derivation rule itself. Therefore, we can use the derivation rules to label the edges of our graphs.

- **Input objects I_i.** Those are the objects provided as input to the tool. All the input objects could be of the same or different type. For example, we may have a collection of documents and a collection of Universities. Each document and each University are in this case *input objects*, but those original objects will belong to two different object types (or collections). We will denote objects as ID_i (Input Document) and IU_j (Input University), respectively depending on the collection they belong to. The main characteristic of our model is that we consider that each type of input object can be a set (unordered) or a list (ordered) of *elements*. For example, a document is a list of paragraphs, or a list of words, a university can be seen as a set of data of different type about the university such as name, department names, etc.

- **Derived Objects D_j.** These objects are simpler (lower level) than the input objects but that are still complex enough to be decomposed again. That is, they can be obtained from an input object or from other derived objects, but they can be decomposed again. Therefore, the model allows for derived objects with different levels of "granularity". That is, from one or more input objects, one could obtain different derived objects with different granularity levels. For example, from an input document ID_i, we can obtain its paragraphs DP_{i_j} and from those paragraphs we can obtain their sentences DS_{i,j_k} or their sentences without stopwords DS'_{i,j_k}. From a university IU_i we could derive its name DN_i or the set of its departments (each department will be a derived object) DD_{i_j}.

- **Atomic Objects** A_k. They are the basic objects of our model. They are obtained by decomposition of input objects I_i or derived objects D_j using derivation rules R_i. But they are atomic, that is, they can not be used to produce new objects by decomposing them. It is mandatory in our model to define at least one type of atomic object and therefore the derivation rule needed to obtain it.

The tool must provide a user interface to allow defining all the object types for each of the three levels, and the derivation rules that applied over input and derived objects produce new objects (with lower granularity). Notice that it is not mandatory to define derived objects. The tool will provide basic algorithms to create the derivation rules, some of them accepting parameters, but the user can provide more complex algorithms to define other rules such as external procedures.

Regarding our multipartite graphs, their basic elements are:

- **Nodes.** Objects of the three different levels (input, derived and atomic) will be the graph nodes. Notice that the relationship between two nodes will usually be associated to the derivation rule that obtains one node from the other.

- **Edges.** The relationships among nodes will be represented by the edges of the graph. Generally graph edges will link each derived or atomic object with the object in the previous level from which it has been derived. Besides those basic edges, other edges are also possible to support some basic relationships such as "order relationships". For example, if we extract the words $AW_{i,j}$ of each document ID_i, the order among words is relevant to reproduce the text. Our tool will automatically create an edge among each atomic object A_{i_j} and the next one $A_{i_{j+1}}$ when they are derived from input object I_i that is a list of elements, such as documents. These links between atomic objects will not appear for sets that do not preserve an order.

Note that the use of a derivation rule over objects of a type to produce objects of a lower level, produce a structure that could be a graph or a tree depending of the definition of the rule. For example, to decompose documents ID_i into words, the rules can extract the list of words in the text (tree) or the vocabulary from the collection of documents (graph). That is, the rule for word extraction could extract each word from the ordered list of words in the document text AWT_{i_j}, linking each document with its words and the words among them by its position in the document. Another possibility would be for the rule to extract the words to the vocabulary of the document text AWV_j eliminating duplicates and linking each word with **all** the documents ID_i where it appears. In this second case the set of words is the vocabulary of the collection of documents, in the first case the set of words of each document represent the text of each document (in this way the graph will have all the document texts).

4 Use cases

The objective of this section is to validate the model proposed in Section3. In order to do so, we explain three currently relevant domains, and the useful rules that, in each case, would decompose and transform the original input objects I_i to obtain some useful derived D_j and atomic A_k objects. We will also present some useful queries that could be directly performed on the basic graph obtained with such decomposition to point out the utility of the multi-partite graph model we propose. We present them in order of difficulty.

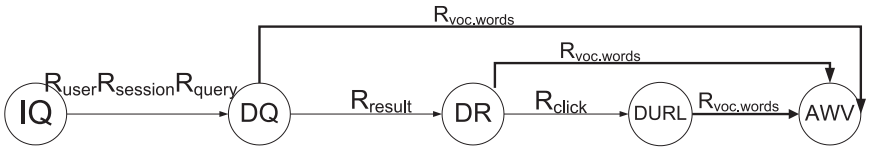

Figure 1: Multi-partite graph model for Query log collections.

4.1 Web Query Logs

Query logs are one of the main data streams that allow search engines to learn from users. Query logs can be used for many different purposes, including query recommendations, spelling correction or even extracting semantic relations [1, 4]. Graphs have been proposed as a good way to exploit query logs for different purposes [2].

Query logs gather all the interaction of users with a search engine, including each query and the clicked pages. Assume that each query log has at least information about users (through anonymized ids), sessions (through cookies), queries, top k results (e.g. $k = 20$), and clicked answers. Then, we can define the following objects:

- IQ_i Each query log in the input data.
- DU_j Each different user in a query log.
- DS_k Each session of a user.
- DQ_ℓ Each query in a session.
- DR_m Each result in the answer to a query, including a Web page (URL), the text snippet and the rank of the result.
- $DURL_p$ Each clicked URL.
- AVW_n Each word in a query, snippet or URL text.

To obtain those derived and atomic objects the rules could be:

- R_{user} Collect all the queries of the same user.
- $R_{session}$ Split the user queries in sessions.
- R_{query} Split the sessions in queries.
- R_{result} Collect all the results of a query in a session.
- R_{click} Extract all the results that were clicked in a given query-session pair. In addition, we could also have the non-clicked results.
- $R_{voc.word}$ Extract all the different words used by a given object, eliminating duplicates and creating the whole vocabulary.

Our derived and atomic objects can be generated using the rules described above as follows:

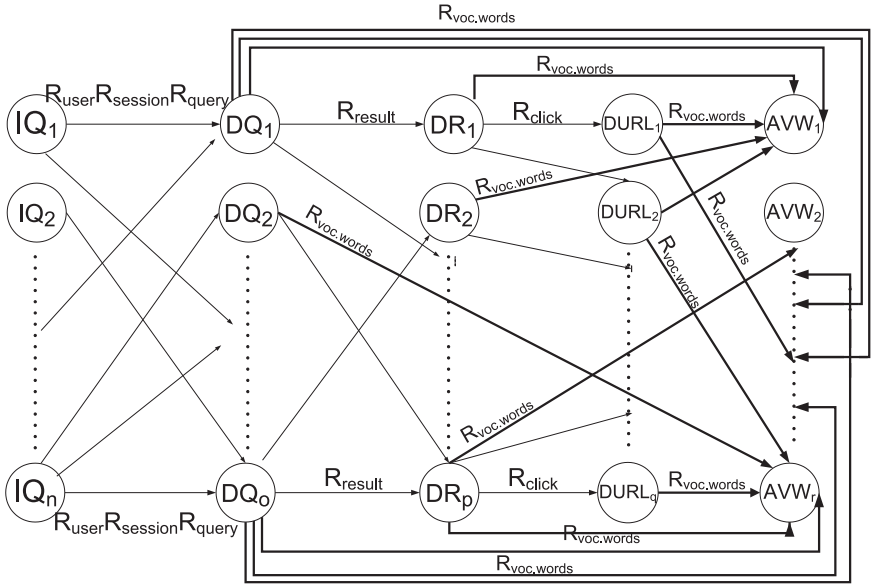

Figure 2: Graph instance for Query log collections.

- $DU_j \leftarrow IQ_i\ R_{user}$
- $DS_k \leftarrow DU_j\ R_{session}$
- $DQ_\ell \leftarrow DS_k\ R_{query}$
- $DR_m \leftarrow DQ_\ell\ R_{result}$
- $DURL_p \leftarrow DR_m\ R_{click}$
- $AVW_n \leftarrow DQ_\ell\ R_{voc.word} \cup DR_m\ R_{voc.word} \cup DURL_p\ R_{voc.word}$

Notice that if we do not need so many derived objects, we could define a sequence of rules, like $DQ_\ell \leftarrow IQ_i\ R_{user}\ R_{session}\ R_{query}$. Figure 1 represent the abstract multi-partite graph model for this last possibility. Notice how only three different derived objects types are defined and that, from all of them the atomic object type AVW is derived with the rule $R_{voc.word}$. Clearly the atomic object typeAVW will be instanced with each one of the vocabulary words.

Figure 2 represents an instance of the previous graph model. The edges representing the derivation rule $R_{voc.word}$ are wider only for clarity.

To use the generated graph a tool for graph management such us [12] is necessary. The tool we are developing will provide the generated graph in GML to make our graphs readable by any such tool, such that useful queries could be efficiently answer. For example it would be possible to know which URL is the most visited counting the number of input edges it has. It is possible to know which queries produce some common results linking them through the result nodes DR and so on.

4.2 Bibliographic Databases

Bibliographic databases are in the center of research and technology progress. They are used to know the relationships among authors, recommend paper reviewers or assess authorities in a

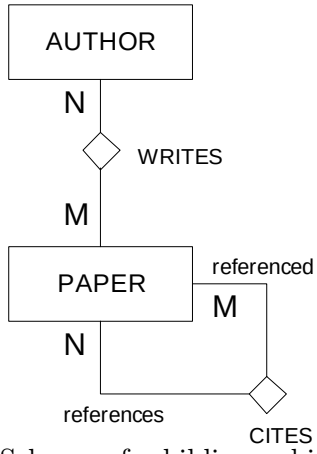

Figure 3: Schema of a bibliographic database.

field [5]. Graphs have been proposed as a way to obtain information that goes beyond the simple list oriented keyword search answers [10].

A bibliographic database gathers all the information that relates authors and papers by means of the relationships among them, i.e. coauthoring relations and citations. Thus, a paper reference is a bibliographic item including the names of the authors and their affiliations and the citations to other papers. A schema of this information is shown in Figure 3.

We consider the following objects:

- IB is the set of bibliographic items.
- DP_i is the derived element paper.
- AA_j are the atomic elements author.
- AC_k are the atomic elements citation.

As it can be deduced from the previous set of objects, authors and citations are atomic objects, and their relationships through papers allow for the creation of a multi-partite graph that relates papers with their authors, and the papers they cite. To obtain those derived and atomic objects, the rules are:

- R_{paper} Collects all the papers from the set of bibliographic items, with a parameter "range years".
- R_{author} Collects the authors of each paper, creating a link with all the papers they coauthored.
- $R_{citation}$ Collects all the citations of a paper, creating a link with the papers that created it.

Our derived and atomic objects can be generated by using the rules in the following way:

- $DP_i \leftarrow IB\ R_{paper}[rangeyears]$.
- $AA_j \leftarrow DP_i\ R_{author}$.
- $AC_k \leftarrow DP_i\ R_{citation}$.

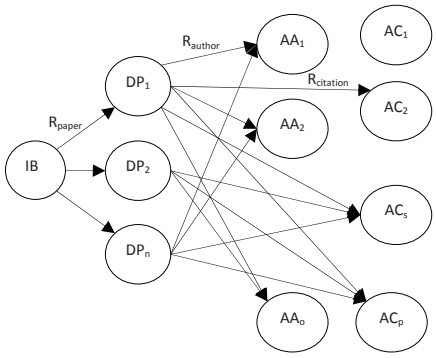

Figure 4: Graph instance for bibliographic collection.

Figure 4 represents an instance of the previous graph model.

However, the complexity of the multi-partite graph that could be generated from a set of references makes any query oriented to a specific set of authors also complex and time consuming.

Thus, for example, a social study of all the authors cited by papers in this century, say between 2000 and 2010, would be difficult to perform if the whole graph was obtained, not to mention the difficulty to generate the graph itself with such an amount of information and its relationships. So in this case the model would help us to reduce the scope of the mining.

Thus, given this example, we could take object IB and apply different rules R_j and the derivative objects to create the multi-partite graph. The rules are: R_{paper} that extract papers written between 2000 and 2010, $R_{citation}$ extracts citations and R_{author} extracts authors from a paper. In the resulting bi-partite graph we can perform queries to create a graph with just relationships among authors.

Now the authors social network can be analyzed by just querying the multi-partite graph.

4.3 Structured Text

Text collections are useful in many application domains where they play different roles from information repository to tool for linguistic analysis. In each case the document collection is used with different algorithms. Our model allows different user rules to define different derived D_j and atomic A_k elements. This way, our model allows for the generation of basic multi-partite graphs from any application domain.

In the following example we assume that we have only one collection of XML documents where authors and title of each document are identified by specific labels. We also assume that each document has only one title but some authors. Lets suppose that those documents have geographic references and pictures with captions that are relevant in the application domain where the collection is going to be used. In this example some of the derived and atomic useful objects could be:

- *Input objects*

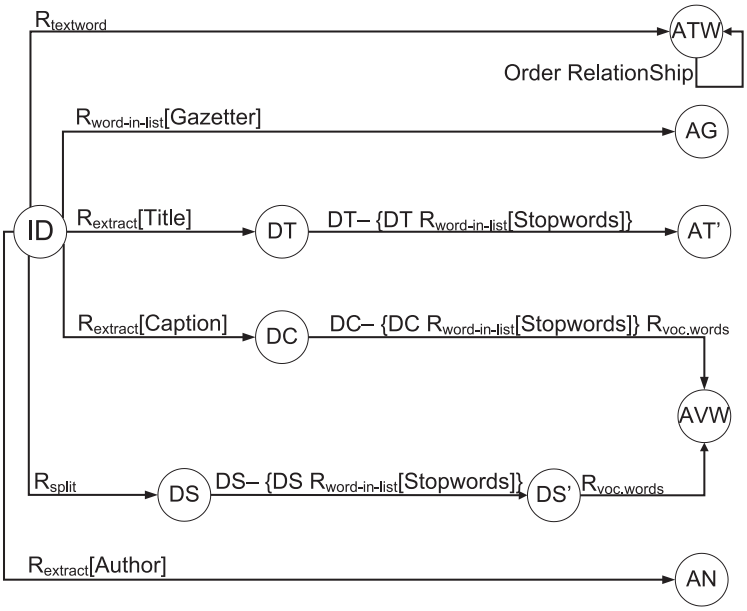

Figure 5: Multi-partite graph model for XML document.

- ID_i Each document in the collection.

- *Derived objects*
  - DT_i Document *Title*. That is, the sentence between the tags <title> < /title>
  - DC_k Each figure *Caption*, that is, sentences between tags <caption> and < /caption>.
  - DS_l Each *Sentence* between two consecutive "periods" of the document (final stops, dots, question mark, etc.).
  - DS'_m Each *Sentence* between two consecutive "periods" without stopwords. It means to check for each word in the sentence if it is in a stopword list.

- *Atomic objects*
  - ATW_{i_k} Each word of each document ID_i, that is, words will appear as many times as they do in the document text, and its order will be preserved to reproduce the text of each document ID_i.
  - AG_n Each toponym in the Gazetteer provided as parameter. That is, words starting with a capital letter that are in a gazetteer
  - AN_v Each author name, that is, sentence between the tags <author> and < /author>
  - AW'_4 Titles without stopwords.
  - AVW_i. Each word that appears in any document. They will just appear once, i.e. duplicates are eliminated. This is the vocabulary of the collection.

To obtain those derived and atomic objects the rules could be:

- $R_{extract}[tag]$ Extract sentences between pairs of tags provided as a parameter.

- R_{split} Cut the document into sentences. That is, it extract each portion of the document between two consecutive period marks.
- $R_{words-in-list}[list]$ Extract the successive words if they are in a specific list provided as parameter (such as stopwords or gazetteers in this example)
- $R_{text.words}$ Obtain the successive word of the text it is applied over.
- $R_{voc.word}$ Extract all the different words used by a given object, eliminating duplicates and creating the whole vocabulary.

Next we show how the derived and atomic objects can be generated using the described rules. We use a basic notation.

- $DT_i \leftarrow ID_i\ R_{extract}[title]$.
- $DC_j \leftarrow ID_i\ R_{extract}[caption]$.
- $DS_j \leftarrow ID_i\ R_{split}$
- $DS'_j \leftarrow DS_i$ - $\{DS_i\ R_{words-in-list}[stopwords]\}$
- $ATW_{i_k} \leftarrow ID_i\ R_{text.words}$
- $AG_k \leftarrow ID_j\ R_{words-in-list}[gazetteer]$
- $AN_k \leftarrow ID_j R_{extract}[author]$.
- $AT'_{j,i} \leftarrow DT_j$ - $\{DT_j R_{words-in-list}[stopwords]\}$
- $AVW_k \leftarrow (DC_i - \{DC_i\ R_{words-in-list}[stopwords]\})R_{voc.words}$
- $AVW_k \leftarrow DS_i\ R_{voc.words}$

As result we will obtain a graph where edges will be between each object and its derived objects of the different types. Notice that authors or toponyms will be linked with all the objects they are related to, but each word AW_{i_j} is only associated to one document ID_i and they also have *order edges* between each word AW_{i_j} and the next $AW_{i_{j+1}}$ in the text they came from. Figure 5 represent the abstract model of the multi-partite graph defined.

In a simpler application it could be possible to have different collections of plain text and the interest could be just to know the vocabulary they have. In this domain maybe only the rule $R_{voc.words}$ for word extraction (eliminating duplicates) would be necessary and the nodes would be only the documents (input objects ID_i) and the words included in the vocabulary of the collection (atomic objects AVW_k), where the set of AVW_k would represent the vocabulary of the collection.

Using any tool for graph management, any instance of the graph model presented in this example (Fig. 5), can be exploited and useful queries can be efficiently answered. For instance, it is possible to know which authors were coauthors in any document using the edges generated by the rule $IDR_{extract}[author]$, that is, we can link any author with any other author through those edges and the nodes for the input object document. In the same way, we can know which toponym appears the most, it requires to see which node AG (remember each one represents a atomic object toponym) has more edges.

This graph model can also be used in more sophisticated applications, for example it would be possible to detect plagiarism looking for sentences (nodes DS), with more than one input edge, that is, sentences that are equal in different input documents. In this way we could obtain documents that have some common sentences. In the same way, using titles without stopwords (nodes AT) we can detect documents speaking about a similar topic.

In all those cases the graph representation of the collections of documents allow for a straightforward analysis of the data that would not be so efficiently performed over the XML document representation.

5 Final Remarks

We have presented a model to generate multi-partite graphs and show their applicability in different data sets. From those graphs, many other graphs can be obtained by applying standard queries over graphs. The next challenge is to design a powerful tool that is simple to use and it is based on our model. A first stage will be through a simple interface, while a second stage will include a graphical interface where nodes and edges will be easily represented.

References

[1] R. Baeza-Yates. Query usage mining in search engines. *Web Mining: Applications and Techniques, Anthony Scime, editor. Idea Group*, 2004.

[2] Ricardo Baeza-Yates. Graphs from search engine queries. In Jan van Leeuwen, Giuseppe F. Italiano, Wiebe van der Hoek, Christoph Meinel, Harald Sack, and Frantisek Plasil, editors, *SOFSEM: Theory and Practice of Computer Science*, volume 4362 of *Lecture Notes in Computer Science*, pages 1–8, Harrachov, Czech Republic, January 2007. Springer.

[3] Ricardo Baeza-Yates, Nieves Brisaboa, and Josep L. Larriba-Pey. Tin2009-14560-c03: High performance processing of large datasets represented as graphs.

[4] Ricardo Baeza-Yates and Alessandro Tiberi. Extracting semantic relations from query logs. In Pavel Berkhin, Rich Caruana, and Xindong Wu, editors, *KDD*, pages 76–85, San Jose, CA, USA, 2007. ACM.

[5] Bibex. http://www.dama.upc.edu/bibex.

[6] Ulrik Brandes and Christian Pich. Graphml transformation. In *Graph Drawing*, pages 89–99, 2004.

[7] Andrei Z. Broder, Ravi Kumar, Farzin Maghoul, Prabhakar Raghavan, Sridhar Rajagopalan, Raymie Stata, Andrew Tomkins, and Janet L. Wiener. Graph structure in the web. *Computer Networks*, 33(1-6):309–320, 2000.

[8] Deepayan Chakrabarti and Christos Faloutsos. Graph mining: Laws, generators, and algorithms. *ACM Comput. Surv.*, 38(1), 2006.

[9] Diane J. Cook and Lawrence B. Holder. *Mining Graph Data*. John Wiley & Sons, 2007.

[10] Sergio Gómez-Villamor, Gerard Soldevila-Miranda, Aleix Giménez-Vañó, Norbert Martínez-Bazan, Victor Muntés-Mulero, and Josep-Lluis Larriba-Pey. Bibex: a bibliographic exploration tool based on the dex graph query engine. In *EDBT*, pages 735–739, 2008.

[11] Semantic Knowledge. http://www.semantic-knowledge.com/tropes.htm.

[12] Norbert Martínez-Bazan, Victor Muntés-Mulero, Sergio Gómez-Villamor, Jordi Nin, Mario-A. Sánchez-Martínez, and Josep-Lluis Larriba-Pey. Dex: high-performance exploration on large graphs for information retrieval. In *CIKM*, pages 573–582, 2007.

[13] Neo4j. http://neo4j.org/.

[14] Álvaro Pereira, Ricardo A. Baeza-Yates, Nivio Ziviani, and Jesus Bisbal. A model for fast web mining prototyping. In *WSDM*, pages 114–123, 2009.

[15] Hanghang Tong, Spiros Papadimitriou, Jimeng Sun, Philip S. Yu, and Christos Faloutsos. Colibri: fast mining of large static and dynamic graphs. In *KDD*, pages 686–694, 2008.