Large Graph Mining: Recent Developments, Challenges and Potential Solutions

Sabri Skhiri and Salim Jouili

Euranova R&D
Rue Emile Francqui. 4, 1435 Mont-St-Guibert, Belgium
{sabri.skhiri,salim.jouili}@euranova.eu
http://www.euranova.eu

Abstract. With the recent growth of the graph-based data, the large graph processing becomes more and more important. In order to explore and to extract knowledge from such data, graph mining methods, like community detection, is a necessity. Although the graph mining is a relatively recent development in the Data Mining domain, it has been studied extensively in different areas (biology, social networks, telecommunications and Internet). The legacy graph processing tools mainly rely on single machine computational capacity, which cannot process large graph with billions of nodes. Therefore, the main challenge of new tools and frameworks lies on the development of new paradigms that are scalable, efficient and flexible. In this paper, we will review the new paradigms of large graph processing and their applications to graph mining domain using the distributed and shared nothing approach used for large data by internet players. The paper will be organized as a walk through different industrial needs in terms of graph mining passing by the existing solutions. Finally, we will expose a set of open research questions linked with serveral new business requirements as the graph data warehouse.

Keywords: Data Mining, Large graphs, Distributed Processing, Business Intelligence

1 Introduction

Data mining is defined variously in the literature of computer science but the common use of the term corresponds to a process of discovering patterns or models for data. The patterns, however, often consist of previously unknown and implicit information and knowledge embedded within a data set [17]. That is, data mining is the process of analyzing data from different perspectives and summarizing it into useful information. In the literature, one can find a large scope of different methods and algorithms that deal with data mining, each of which has its own advantages and suitable application domains [24, 44, 31]. Those techniques have been heavily developed these last years in Business intelligence [50, 41] especially for database and flat data in order to feed market analysis, business management, and assisted-decision tools [17]. It is worth saying that the data mining stands at the intersection of different disciplines such as statistics, machine learning, information retrieval and pattern recognition. Almost all mining algorithms can be divided into the following families: (1) the classification for which we position data in pre-determined groups, (2) clustering in which data are grouped within partitions according to different criteria, (3) associations that enables to link data between each other, (4) pattern recognition in which we mine data to retrieve pre-determined pattern, (5) feature extraction and (6) Summarization (Ranking such as *PageRank*).

Traditionally, the algorithms manage and process the data as a collection of independent instances of a single relation. That is, the instances of data to be mined are considered independent without relationships between them. For example, in the case of the clustering algorithm in which the input data set is divided in groups with similar objects, it is considered that there is no relation between the objects. Hence almost all clustering algorithms compute the similarity between all the pair of objects in the data set by means of a distance measure. Indeed, the traditional data mining works are focused on multi-dimensional and text data. However, nowadays new emergent industrial needs lead to deal with structured, heterogeneous data instead of traditional multi-dimensional models. This kind of structured dataset is well designed as graph that models a set of objects that can be linked in a numerous ways. The greater expressive power of the graph encourages their use in extremely diverse domains. For example, in biology, the biochemical networks such as the metabolic pathways and the genetic regulation known as the transduction signal networks constitute a significant graph of interactions. On the other hand, the graphs are used in chemical data processing in a way that the molecule structure is described as a graph which implies that the molecule catalogs are processed as graph set. In the Internet area, the rise of social networks shown the need to model the social interactions as graphs. We can find another example in credit card fraud detection in which transactions are modeled as a bipartite graph of users and vendors.

This modeling change involves a paradigm shift in the way to apply the mining algorithms. We need to ensure that the classical data mining techniques are still equally applicable on graph models. The problem that arises here results from the fact that almost all needed measures such as similarity and distance cannot be easily defined for graph in as intuitive way as is the case for multidimensional data. As a matter of fact, the mining algorithms for graph are more challenging to implement because of the structural nature of the data. The second challenge that arises in many applications of graph mining is that the graphs are usually very large scale in nature. Indeed, in practice, those graphs can reach a significant size, as in social networks or interaction graphs. This kind of graphs can typically reach several hundred millions of nodes and billions of edges. However, most of the graph mining algorithms deal with data already available in main memory which is not accurate in large scale graph.

These last years we have seen new techniques emerging for facing this kind of large graph processing: (1) high performance graph DB such as DEX [55] or Titan [64], (2) in-memory and HPC/MPI graph processing such as SNAP [5, 6] and finally (3) the distributed approach based on Bulk Synchronous Processing such as Pregel [51]. As a result graph miners will face three important issues: (1) adapting the mining algorithms to make them graph-aware, (2) redesigning the algorithms to be implemented by those new high performance techniques, (3) storing and exploiting many different graphs and to be able to apply similar processing as in traditional data warehouses.

In this paper we will focus on the distributed processing approach and we will show how this new generation of distributed graph processing frameworks can be used to implement typical graph mining algorithms. The second half of the paper will describe whether having a high performance data mining stack is a sufficient condition to get a graph data warehouse stack.

Section 2, Graph mining algorithms, will present well-known data mining algorithms in clustering and classification areas. The section "distributed graph processing framework" will introduce new emerging distributed graph processing frameworks and will describe how the algorithms previously exposed can be implemented on such frameworks. Finally, the section "Graph Data Warehouse: an emerging challenge" extends the concept of the graph mining to graph data warehouse processing and describes the new challenges that must be tackled by the research communities in order to reach the same level of performance as existing relational data warehouses.

2 Graph mining algorithms

The objective of this section is to introduce typical graph mining algorithms that will be discussed in the next sections for their distributed implementations on the Pregel paradigm. We first introduce a traditional graph mining algorithm used for large graphs, *PageRank*, and then we present a typical example of an existing mining algorithm for clustering that must be adapted to be graph-aware and fully leverage the linkage information.

2.1 Ranking: *PageRank*

The world wide web structure can be seen as a graph in which the web pages are the vertices and the (hyper-)links are the edges. However, in addition to the web page contents, the graph structure of the web presents a very important additional source of information which can hold implicit knowledge about the web pages. Since the nineties, some works have focalized their efforts on how to exploit this topological structure of world wide web (see, e.g., [11, 12, 14, 17, 20, 42, 43, 54, 59]). One of the most famous works which exploits the topological structure of the web is the *PageRank* algorithm [14, 59]. This algorithm has been stated as one of the key to success of the well-known Google search engine[1] [2].

The *PageRank* algorithm computes a ranking for every web page based only on the linkage structure of the world wide web (graph of the web). The authors

[1] `www.google.com`

of *PageRank* introduce the notion of page authority, which is independent of the page content. In *PageRank* algorithm, the authority is approximated from the number and importance of the pages pointing to the involved page. This algorithm considers a page as *"important"* if it has many incoming pages and/or it has a few highly ranked incoming pages. That is, a page has high authority (rank) if the sum of authorities (ranks) of its incoming pages is high.

The *PageRank* of pages is computed by following a random surfer which browses the web from page to page. The random surfer is a random walk such that the set of states is the set of Web graph vertices, and at each random step, with some probabilities, (1) the surfer chooses an outgoing link of the current vertex uniformly at random, and follow that link to the destination vertex, or (2) it *"teleports"*[2] to a completely random Web page, independent of the links out of the current vertex. Intuitively, the random surfer traverses frequently *"important"* vertices with many vertices pointing to it.

Let $\mathbb{G} = (V, E)$ be the web graph with vertex set V and edge set E. Let $\mathbf{d}_{out}(v)$ be the number of outgoing edges from the vertex $v \in V$. Let $\mathbf{d}_{in}(v)$ be the number of incoming edges to the vertex $v \in V$, i.e., the in-degre of v. Let p, $(0 < p < 1)$, be the damping factor (usually set to 0.85) that represents the probability with which the surfer follows with the random walk, while $1 - p$ is the probability of teleporting to a random vertex among all $|V|$ vertices. Thus, the *PageRank* $\mathbf{PR}(v)$ of vertex (page) v is given by the following formula [59]:

$$\mathbf{PR}(v) = \frac{(1-p)}{|V|} + p \times \sum_{u \in \mathbf{d}_{in}(v)} \frac{\mathbf{PR}(u)}{\mathbf{d}_{out}(u)} \tag{1}$$

In a matrix form, Equation 1 can be rewritten as:

$$\mathbf{R} = p \times (\mathbf{AR} + D) \tag{2}$$

where the matrix $\mathbf{A}$ is a square matrix with the rows and columns corresponding to graph vertices, $\mathbf{A}_{u,v} = \frac{1}{\mathbf{d}_{out}(u)}$ if there is an edge from u to v and $\mathbf{A}_{u,v} = 0$ if not, $\mathbf{R}$ is a column vector representing the ranks of pages, and D is a constant vector $(= (1-p)/|V|)$.

While we will not go deeper into the mathematical underpinnings of *PageRank* here, it is shown that Equation 2 can be resolved with an iterative solution (see Equation 3) that converges for $0 < p < 1$:

$$\mathbf{R}_{i+1} = p \times (\mathbf{AR}_i + D). \tag{3}$$

2.2 Graph clustering

Clustering data is a fundamental task and one of the most studied topics in data mining [35, 39]. Given a set of data instances, the goal is to group them

[2] *Teleportation* step: choose a vertex uniformly at random, and jump to it. This step is needed because it exists some vertices that does not have outgoing links (non-ergodic graph)

into groups that share common characteristics based on similarity. Intuitively, instances within a cluster are more similar to each other than they are to an instance belonging to different clusters. In the context of graph data, the clustering task is usually referred to as communities detection within graph [69, 56, 21, 19, 62, 29]. In the case of a citation graph of the scientific literature, the vertices correspond to the papers and the edges correspond to citation relationships[3]. By clustering this graph, for a given paper one can identify the community of surrounding relevant works, without traversing the cited works nor the works they cite as well.

Here, we describe two graph clustering algorithms: the first algorithm is a generalization of the well-known k-means [50] algorithm and the second is a divisive algorithm which uses a structural-based index to gather information about the separable communities in a graph.

***k*-means based clustering** In multidimensional data context, the k-means [50] algorithm is the simplest and one of the popular algorithms used for clustering. It minimizes the sum of the distances between the data instances and the corresponding centroids. The k-means [50] algorithm needs two parameters : (1) k the number of groups to provide and (2) $D : (o_i, o_j) \rightarrow \mathcal{R}$ a distance measure that maps pairs of data instances to a real value, it works as follows:

1. Randomly selects k data instances as the initial cluster centers ("centroids").
2. Each data instance in the dataset is assigned to the nearest cluster, based on the distance (computed by D) between each one and each cluster center.
3. Each cluster center is recomputed as the average of the data instances in that cluster.
4. Steps 2 and 3 are repeated until the clusters converge. Convergence may be defined differently depending upon the implementation, but it normally means that either no objects change clusters when steps 2 and 3 are repeated or that the changes do not make a material difference in the definition of the clusters.

The k-means algorithm has been extended recently to allow its use in linkage structure graph. In order to achieve reliable and efficient generalization of k-means to graph domain, two key issues have been addressed. Firstly, the distance measure D has been defined in such way it takes into account the relationship between vertices. Intuitively, the distance between two vertices is considered as the geodesic distance which is the number of edges ("hops") in a shortest path connecting the pair of vertices in question. Secondly, the computation of the cluster centers ("centroids") requires graph-aware procedures that can efficiently select a vertex which is the most representative of a set of vertices. One possible solution consists of using the notion of *median vertex* which is a vertex that minimizes the sum of distances to all the other vertices. Formally, let C be a set of vertices and D a distance measure for the graph, the median vertex $\widehat{v}$ of the

[3] Two papers are connected by edge if one cites the other

set C is defined as follows:

$$\widehat{v} = \underset{v \in C}{argmin} \sum_{u \in C} D(v,u). \tag{4}$$

Besides the median vertex, Rattigan et al. [62] uses the closeness centrality [27, 72] to select the representative vertex of a cluster, i.e. they select the vertex with the greatest closeness score. A vertex will be consider in a central position according to his closeness score. This measure stresses that the quality (position in the graph) is more prominent than quantity (number of incident edges). Formally, the closeness centrality measure of a vertex v in a cluster C is computed as follows:

$$CC(v) = \frac{|V| - 1}{\sum_{v \neq u, u \in C} D(v,u)} \tag{5}$$

where $d(u, v)$ is a distance measure for the graph and $|V|$ the size of the graph.

As it can be remarked, one of the most important and common task in the graph clustering (and other graph algorithms) is the distance computation between vertex. As aforementioned, the shortest path computation forms the most used way to compute such distance. Intuitively, the goal is to find the shortest path from a source vertex v to a target vertex u, among all paths that satisfy a certain criterion. The applications of the shortest path computation cover a large scope of computer science fields, including network optimization [7, 48], scheduling [28], image processing [57], geographic information systems [73], social network [75]. In the literature, since the late 50's, shortest path problem was very well studied and many solutions to this problem have been proposed (i.e. see [68]). However, the earlier algorithms [9, 22, 26] are still used nowadays, especially Dijkstra's algorithm [22]. Dijkstra's algorithm solves the single-source shortest-path problem when all edges have nonnegative weights. It starts at a source vertex and explores the entire graph in all directions until the distances to all the other vertices (reachable from the source) are computed. Dijkstra's algorithm is considered as a greedy algorithm because it selects, in each step of the traversal process, the local optimum, which is the edge that satisfies the considered criterion.

Centrality based clustering Before detailing this part, we provide a definition of one of the numerous centrality measures that is used for graph clustering. Here, we focus on the edge betweenness centrality [72, 27] which locates, structurally (content-independent), the *well-connected* edges within a network. Here, an edge is considered to be well-connected if it is located on many shortest paths between pairs of vertices. That is, an edge with high betweenness centrality performs, in some sense, a control over the interactions between vertices. For instance, if two non-adjacent vertices v and w seek to communicate (interact) and their shortest path pass through an edge e, then e may have some control over the communication between v and w. This betweenness centrality of an edge e is calculated as follows:

$$BC(e) = \sum_{v,w \in V} \frac{b_{vw}(e)}{b_{vw}} \tag{6}$$

where $b_{vw}(e)$ the number of shortest paths from v to w that pass through e and b_{vw} the number of all shortest path between v and w.

Girvan and Newman [29] propose a divisive method that is well-suited for the social context. The divisive method starts with the whole graph and iteratively cuts specific edges. This divide the graph progressively into smaller and smaller disjoint subgraphs (communities). The key problem for these methods is the selection of the edges to be cut. Indeed, the edges to be cut should connect, as much as possible, vertices in different communities and not those within the same communities. Girvan and Newman [29] use edge betweenness centrality to select edges to be cut, for connected graphs. The idea behind this is that the inter-community edges have high betweenness centrality, while the intra-community edges have low betweenness centrality. The proposed algorithm works iteratively in two steps:

1. Compute the betweenness of all existing edges
2. Remove the edge with highest betweenness centrality
3. Repeat Step 1 & 2 until the communities are suitably found

The stopping criteria can be designed by a *"a priori"* definition of *"suitable communities"*, and at each iteration we test whether the resulting subgraphs fulfill the definition. It is worth saying that this algorithm is very useful for web graph and social graph because they are characterized by small-world structure property [46, 44].

Summarizing, it is clear that the graph clustering is a challenging topic. This stems, firstly, from the fact that the clustering algorithms can be useful for some graph types and not for others. Secondly, they are almost all computationally expensive because they need to re-compute several measures in each step (e.g. the betweenness for each edge). Nevertheless, nowadays, the real world applications need to deal with very large-scale graphs such as social networks or web graphs where the size grows exponentially (billions of vertices). For sake of scalability, some works try to provide a faster clustering solution by considering only local quantities, such as [61] but this is still not sufficient for real world applications. Yet, a major part of graph clustering algorithms seems to discard the evolving aspect of the real graphs. Indeed, the structure of graph in almost all domains changes over time by adding/removing edges and/or vertices.

3 Distributed graph processing framework

3.1 Distributed computation framework

Parallel and distributed computing have been strongly studied these last twenty years and they have been tremendously popularized by new frameworks such as

MapReduce [45, 25, 77, 47, 13] (see §3.2) and Dryad [34, 60]. The notion of parallelism represents the ability to run simultaneously software in different processors in order to increase its performance while the distributed concept emphasizes the notion of loose coupling between those processors. The distributed architectures can be described and classified according to the resources that the machines or the processors share with each other. This classification is particularly important if we speak about large graph storage and processing.

The main categories are shared-memory, shared-disk and share-nothing. The shared-memory architecture describes distributed systems that share a common memory space. In the case of distributed machines, it can be a distributed cache where the data is commonly available. In large-scale super-computer the processors can share the data through non-uniform Memory Access (NUMA)[36]. However, this kind of architectures is more suited for small parallel data problem, as the shared memory must manage the data consistency and accesses through the different clients. The second category is the shared-disk that enables to connect distributed processors by Local Area Network (LAN). Even if they are less costly in term of scalability when adding new storage nodes than the shared-memory, they still suffer from the access contention and the data consistency when number of processor clients dramatically increases. Finally the last category is the shared-nothing architecture in which each machine has its own independent storage. Therefore, a new important concept has been defined, *the partitioning*. The data are partitioned over the cluster of machines according to a partitioning policy. This policy defines the location of the data and then, the distributed computing framework can send dedicated tasks where the data is located. This represents the notion of data *locality*.

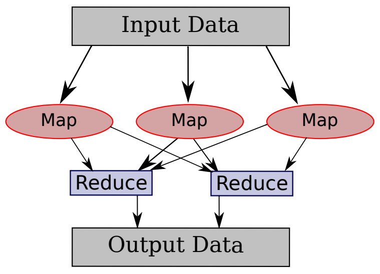
The parallel programming paradigm can be described as either explicit or implicit parallel programming. In the explicit version, the developer will have to explicitly create tasks, synchronization points, managing threads and processes and ensuring that the parallel operations will be safe, etc. On the other hand, in the implicit parallel programming, the developer does not need to care about these details. The compiler or the distributed framework handles all aspects related to the parallel execution such as the portion of the code that must run in parallel, the task location routing, the creation of threads and processes, the data access, etc. Although the explicit parallel programming is richer and let the developer accurately drive the distributed processing, it represents a serious complexity in term of design and implementation, and is error-prone. Most of the distributed processing frameworks presented in this paper are shared-nothing and expose an implicit programming model.

3.2 Large graph processing

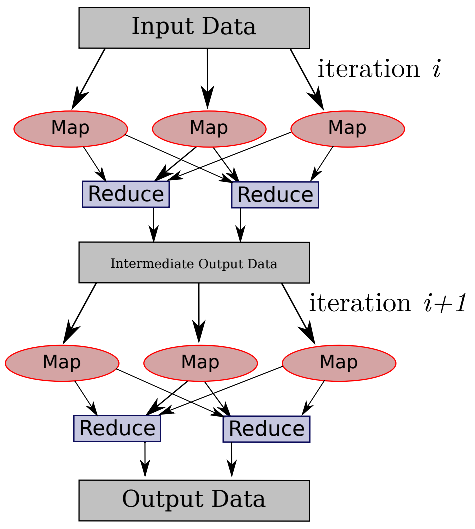
The MapReduce technique, proposed by Google, is a famous paradigm in distributed computing on large data sets and can be used for computer programs that need to process and generate large amounts of data. For instance, MapReduce has been used by Google to create the index of all the crawled web pages. Hadoop [10] is an open-source implementation of MapReduce. In addition to the distributed computing, the three main strengths of Hadoop result in data locality, fault tolerance and parallel processing. For the sake of completeness, we define briefly the two major steps of the MapReduce paradigm:

– **Map**: in this step, the problem is partitioned into a set of small sub-problems. This set is then distributed over the machines available in the cluster and each sub-problem is processed, independentely, in a single machine, namely worker node.
– **Reduce**: in this step, all the answers to all sub-problems are gathered from the worker nodes and are then merged to form the output solution.

These two functions are written by the user and are applied to distributed data over a cluster of machines as shown in Figure 1(a). This programming model offers to developers an easy and simple way to deal with large data sets in a distributed computation environment. Indeed, by means of Hadoop, the developers do not need to be experts on distributed computation, and they have just to focus on the design of their algorithms with MapReduce programming paradigm. Nevertheless, this paradigm is not well suited for graph processing tasks and iterative algorithms. In fact, a simple iterative algorithm needs a whole execution of a MapReduce task in each iteration which forms a huge data migration and computation over the algorithm execution, requiring lots of I/Os and unnecessary computations. Figure 1(b) illustrates two iterations with a naive implementation with MapReduce paradigm (in each iteration the data are partitioned, processed and the intermediate results are stored).



(a) MapReduce programing model



(b) Naive iterations with Mapreduce

Fig. 1. MapReduce paradigm and a naive iteration implementation.

In order to overcome this problem, some works [18, 23, 15, 37, 38, 51, 76] proposed a set of techniques to improve classical MapReduce for iterative algorithms. For instance, Twister [23] and Haloop [15] solutions reuse workers across iterations by changing only the input which minimizes the number of instantiated workers. In addition, Haloop [15] supports caching of input and output of iterations to save I/Os and it uses a loop-aware scheduling of jobs which is a very interesting extension of Hadoop. Despite the improvements, these solutions lack efficiency for the graph-based algorithms since they deal essentially with multidimensional data. In this respect, some methods have been developed to deal especially with the distributed computation of linkage structural data [51, 18]. In this context, the most popular framework was introduced by Google and called Pregel [51]. The main purpose of Pregel is to provide a distributed computation framework entirely dedicated to graph processing algorithms implementation. Google Pregel was inspired from the Bulk Synchronous Parallel (BSP) programming paradigm [71]. Roughly, in the BSP model an algorithm is executed as a sequence of *supersteps* separated by a global synchronization points until termination. Within each superstep a processor (or a virtual processor) may perform the following operations; (1) perform computations on a set of local data (only) and (2) send or receive messages. Similarly, in Pregel, within a *superstep* the vertices of graph execute the same user-defined function, in parallel. This function can include : a modification of the state of a vertex or that of its outgoing edges, read messages sent to the vertex in the previous superstep, send messages to other vertices that will be received in the next superstep, or even a modification of the topology of the graph (deleting or adding vertices and/or edges) [51]. Pregel uses a *"vertex voting to halt"* technique to determine the algorithm termination. Each vertex has two possible states: *active* or *inactive*. An algorithm is considered terminated when all the vertices are in the *inactive* state. Practically, in the initial superstep (superstep 0), all vertices are in the *active* state, then in each subsequent superstep each vertex can vote to halt and then, explicitly deactivate itself. An *inactive* vertex do not participate on any superstep unless it receives an non-empty message[4].

There exists several open-source implementations of Pregel, but the two mature ones are Apache Hama and Apache Giraph. In the remaining of section, we will address the implementation of two graph algorithms with Pregel paradigm: single source shortest path (SSSP) and pageRank.

[4] The fact of receiving a message activates the vertex state

```
**Input** : Messages: Set of received messages
if currentVertex is the source then
    minimumDistance ← 0;
else
    minimumDistance ← $\infty$;
end
foreach message $m \in$ Messages do
    minimumDistance ← minimum(minimumDistance, valueOf($m$));
end
if minimumDistance $\leq$ valueOf(currentVertex) then
    valueOf(currentVertex)← minimumDistance;
    foreach outgoing edge $e$ from currentVertex do
        sendMessageTo(targetOf($e$), (minimumDistance+ valueOf($e$)));
    end
end
VoteTohalt();
```

Algorithm 1: Pregel: vertex function for Single source shortest path problem.

SSSP implementation: Algorithm 1 shows a pseudo-code of the vertex function for a SSSP implementation within Pregel framework. Initially, each vertex value (except the source), which corresponds to the distance to reach it from the source, is initialized to an infinity constant (larger value than any possible distance in the graph). In this algorithm, in each superstep, each vertex reads messages from its neighbors. Each message contains the distance between the source vertex and the current vertex (through a given adjacent vertex). For a given vertex, if the minimum message value is less than the actual associated value, the vertex updates its current value. Then, the vertex sends messages through all its outgoing edges, such that each message contains the sum of the weight of its outgoing edge and the new value associated to the vertex. Finally, the vertex vote to halt. The algorithm terminates if there is no more updates performed. As result of the algorithm, each vertex of the graph will be associated to a value that denotes its minimum distance from the source vertex to it. In the case of unreachable vertex from the source (unconnected graph), the associated value to such vertex is set to the infinity constant.

For sake of more demystification, let us analyze an example of the execution of the previous algorithm. For this purpose, Figure 2 provides a superstep by superstep execution of the SSSP algorithm on a sample graph. Here, we consider the vertex labeled by (1) as the source. The initial step consists on setting the values associated to all the other vertices to infinity. In superstep 1, the vertices (2), (3) and (4) receive from the vertex (1) (in superstep 0), respectively, the messages containing their distances to (1). For instance, the vertex (2) receives a message that contains 6 which is the sum of the value of vertex (1) and the weight of outgoing edge ((1)→(2)). Moreover, in superstep 1, the source vertex is in inactive state because it does not receive any message in this superstep. The next supersteps follow the same procedure until all the vertices are in inactive state.

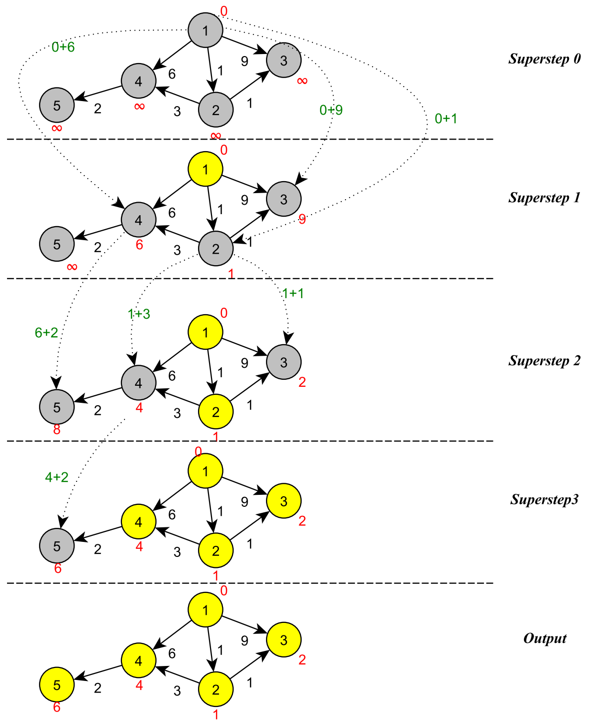


Fig. 2. Single source shortest path supersteps. Dotted lines describe how the messages are sent (from/to), the green labels are the message values. Yellow vertices have voted to halt.

***PageRank* implementation:** Algorithm 2 shows a pseudo-code of the vertex function for a *PageRank* implementation within Pregel framework. Initially, each vertex value, which correspond to the *PageRank* estimation, is initialized to $\frac{1}{SizeOfGraph}$. In this algorithm, in each superstep, each vertex read messages from its neighbors. Each message contains tentative pageRank divided by the number of outgoing edges of the involved vertex. For a given vertex, the received message values are summed up into *sum*, then the vertex updates its current *PageRank* value by $\frac{0.15}{SizeOfGraph} + 0.85 \times sum$ (which follows Eq. 1). Then, the vertex sends messages through all its outgoing edges. Finally, after a fixed number of supersteps (iterations) the vertex vote to halt. The algorithm terminates if there is no more updates performed. As result of the algorithm, each vertex of the graph will be associated to a value that denotes its *PageRank*. As mentioned in [51], instead of fixing the number of iteration, one could find a suitable setup of this algorithm to run until convergence of *PageRank* values.

```
Input   : Messages: Set of received messages
if NumberOfSuperstep $\geq$ 1 then
    sum $\leftarrow$ 0;
    foreach message $m \in$ Messages do
        sum $\leftarrow$ sum + valueOf($m$);
    end
    valueOf(currentVertex)$\leftarrow$ 0.15 / SizeOfGraph + 0.85 $\times$ sum;
end
if NumberOfSuperstep < MaximumNumberOfIteration then
    N = SizeOf({outgoing edges from currentVertex})
    sendMessageToAllNeighbors(valueOf(currentVertex) / N);
else
    VoteTohalt();
end
```

Algorithm 2: Pregel: vertex function for *PageRank*.

4 Graph Data warehouse: an emerging challenge

Since we can provide efficient methods for processing and mining large graph, the next question to answer is how to store many of these large graphs and still exploiting their informational potential. Nowadays, we end up with a significant number of graphs in data warehouse, not because they are the easiest way to analyze the data but because they are the most meaningful way to represent the relationship concepts.

The concept of graph data warehouse is similar to the traditional data warehouse: the warehouse is fed by a set of data sources that can be either databases or existing graphs. Those data sources are extracted and transformed to be loaded as a graph structure in order to facilitate the data mining. The problem is then (1) how to merge different graphs from different data sources, (2) how to define an efficient conceptual modeling on top of graph data and finally, (3) how to express efficient queries. Let us take the example of a Telecom operator who owns (1) a network address book service, (2) a chat service, (3) a social network, and (4) a set of call data records. The objective of this Telco is to merge that information within on single aggregated graph in order to extract and infer information such as: influencers and maven, potential churners and service usage pattern. Therefore the first question is how to merge the different graphs from the address book, the chat service contacts, the social network and the CDRs, knowing they are different services launched by different departments and using different ID for users. The second question would be how to define a conceptual model enabling to define roles in edges we can navigate, additivity for the time spent on each service, the navigation path on edge for clustering algorithm that can be used for maven identification, etc. Finally, we have the queries such as once the maven have been identified, retrieve the closest users based on their location or retrieve the average time spent on all services by influencers or potential churners. We can also consider composite queries such as (1) extracting all male influencers living in cities of more than 1M inhabitants, (2) extracting potential churners who are in the neighborhood of those influencers and (3) extract last year churner in the graph neighborhood of the current potential churner and evaluate their interaction through the social network service and CDR, (4) according to the query (2) and (3) defining the best influencers to propagate a promotional message.

This example leads to the key question of this section: if the graph modeling enables to better leverage the relationship concept in mining algorithms and if several execution engines and distributed frameworks enable to apply those mining algorithms on significant graphs, what is still missing for a graph data warehouse? This section will answer to this question by first introducing basic foundation of data warehouse approach, from there we will explain the existing lack of methodologies, modeling approaches and analytic tools for an equally efficient graph data warehouse.

4.1 Why not using relational data warehouse for storing graphs?

The data mining algorithms are involved in each step of the traditional data warehouse [33], we can use techniques for identifying key attributes or finding related measures or dimensions, some other techniques enable to limit the scope of the data extraction on specific clusters. Usually the OLAP framework is integrated with a Mining framework in order to use both in conjunction, this integration is often named On-Line Analytic Mining (OLAM) and exploratory multi-dimensional data mining [31]. According to [31], there is at least four ways to use OLAM and OLAP in conjunction:

- using the multi-dimensional cube space for defining the data space for mining
- using OLAP queries for generating features and targets for mining
- using data mining as building blocs in a multi-step mining process
- using data cube computation for speeding-up repeated models construction

The graph model should be another way to represent the information and then it should be stored as any other data sources in the data warehouse. However, the graph model must be considered as a constraint. Indeed, if it is represented as a graph it is because it leverages the relationship concept. Let us take the example of a social network. We could represent it as traditional relational tables on which we can represent the concept of user as a table pointing to a m-n table defining all the relationship a user has. If we consider that, in average, in a social network a user has 100 friends, a simple request retrieving the friends of the friends will lead to 100^2 join requests. A social network can be stored without any issues on a relational data warehouse, but when comes the question about how we can leverage the relationship concept through a set of mining algorithms, we come to the conclusion that the graph model remains the most appropriate format. In addition, most of the mining techniques must navigate through the edge relations. As shown by Figure 3, this means that it (1) will significantly cost in terms of join operations and (2) will almost transfer the totality of the graph between the relational storage and the mining application, which could potentially represents a huge amount of data in the case of the graphs we consider in this paper.

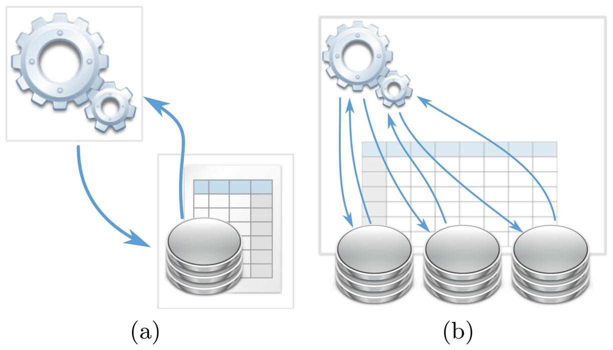


Fig. 3. (a) The traditional relational DB approach involves that almost all the graph content would be transferred between the mining application and the storage. (b) While in a data application server concept the mining application is implemented as an application in the distributed storage midlleware.

We saw in the previous section how we can use new emerging distributed frameworks in order to apply traditional mining algorithms on significant graphs. This means that graph mining analysis such as classification, link-based analysis, object or link-based recommendation, trust computation can be applied on graph models. An interesting advantage of those frameworks is that they can both leverage the data locality, e.g., the information about the location of sharded data, and they can integrate mining algorithms within the distributed storage. As a result they highly minimize the quantity of data exchanged between the processing and the storage nodes during a mining operation.

4.2 Traditional data warehouse approaches

For many years the knowledge presents in the data accumulated by an enterprise has always represented a key strategic element in its management, in term of business KPI, behavioral analysis, strategic marketing and many other fields. The traditional data warehouse aims at providing the software, the modeling approaches and the tool to analyze the set of data present in a collection a databases.

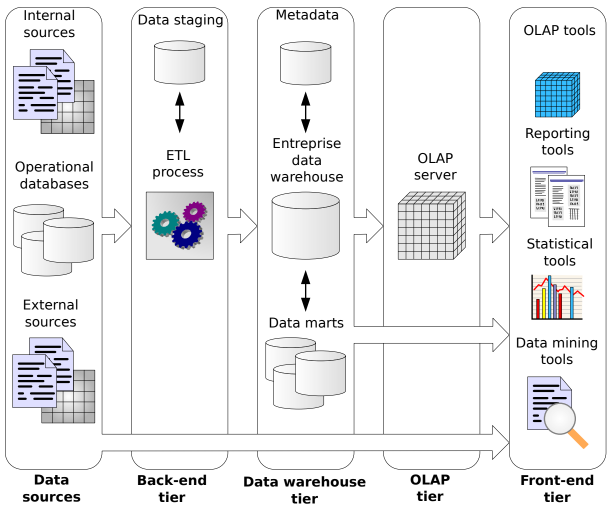


Fig. 4. A traditional process overview [52].

Figure 4 shows the traditional process to extract knowledge from data. A first step consists in extracting data from the collection of data sources that can be relational, files, remote web sources, etc. The ETL (Extract, Transform and Load) aims at structuring the extracted data in a more processable form, this is the data warehouse tier. This phase is usually done manually by using an ETL software, without specific modeling approach. However new emerging researches try to adopt a complete modeling process aligned with the modeling techniques used in the OLAP tier [3].

The data warehouse tier is then used for designing high level models that will be used for executing specific requests. This is traditionally the area of On-Line Analytical Processing (OLAP). The idea is to design a logical model suited for regrouping the data that are needed for the OLAP queries and to generate a related physical model. The OLAP queries take advantage of the physical model and the query model to generate a physical execution plan. The development of conceptual modeling of data warehouse has been an important research area that is still highly active. Most of the research topics focus on the improvement of the *snowflake* and *star* schema [53]. Some researches try to add a graphical representation [63] based on the ER model [65, 70] or based on UML [1, 49], other focus on models that enable to define different level of hierarchies [63, 8, 32, 40], while others provide models that take into account the role played by a measure in different dimensions [49, 41]. The model described by [53] tries to summarize the main limitations of the snowflake and star model and proposes a new model that includes most of the previous researches in this area.

The last phase of the processing is the OLAP tier in which the data processing is led by the expression of the OLAP queries. An overview of OLAP techniques has been described by [16]. The authors describe the main OLAP operations as rollup (increasing the level of aggregation) and drill-down (decreasing the level of aggregation or increasing detail) along one or more dimension hierarchies, slice and dice (selection and projection), and pivot (re-orienting the multidimensional view of data). Usually the OLAP queries are expressed using standard SQL or Multidimensional Expression language (Mdx) from Microsoft [58].

In conclusion we can summarize the complete data warehouse process by (1) the storage and (2) the process to extract, (3) to model and (4) to query the data. In the next section we will show that graph data warehouse is far from the maturity reached by the legacy warehouses.

4.3 Graph data warehouse challenges

In previous sections we have shown that distributed processing frameworks can be used to implement graph mining algorithms. However, this area lacks dramatically an unified conceptual approach to mine graph as found in legacy data warehouses. Figure 5 shows an equally functional data warehouse process when dealing with graphs. Similarly the data sources can be existing databases, as it is the case in fraud detection or call data records in telecommunication or even existing graphs. After an ETL process, we end-up with a consolidated graph that is the integration of different graphs. This graph must be queried and analyzed. Currently, there is no common approach that enables to model an equally functional conceptual modeling approach such as the multi-dimensional cube for graphs. We need to be able to model an intermediate structure keeping the relationship as a central concept but enabling to represent different navigation paths, different roles in those paths while being able to represent hierarchies. There is a real gap in the research community in this area although the need for this kind of conceptual modeling approach will grow in the coming years. However, existing conceptual modeling approaches such as the Multidim model [53] should be independent of the underlying physical model, but it will need to be extended for specific graph semantics such as navigation paths, role in relationships and other properties such as the additivity in the navigation path. In the same way, there is no graph query language giving the same level of flexibility as existing OLAP queries. Although few graph languages exist, they have been designed with specific objectives in mind and do not really represent an equally functional OLAP query language. We can cite Gremlin [4] that comes from the graphDB area and SparQL [67, 30] which defines standard query language and data access protocol, mainly used for RDF meta-models. Finally, the graph mining and data warehouses need to have an integrated execution processing framework. As in legacy OLAP, we need to generate, from the graph query, a logical execution plan and finally, a physical execution plan taking advantage of the distributed aspects of the graph. Currently, there is no way to express a graph query and to generate a distributed execution plan as it can be found in Apache PigLatin. However, few researches especially in the web semantic try to leverage existing distributed processing frameworks such as MapReduce [66], but it is highly dependent to RDF and does not really leverage the graph aspects. As for the conceptual modeling, this is clearly a gap in the research community.

It is worth noting that, at the moment of writing this paper and at our level of knowledge, the most complete research project on graph data warehouse is GraphCube[74]. The authors defined the concept of multidimensional network which is a graph on which each vertex is a tuple in a table. The attributes of this table represent the multidimensional space. The authors showed that we can execute the same OLAP queries on a table and a corresponding graph. This work enables to mine in the same time traditional relational sources and a graph as an additional source of information. They also defined the algorithms to obtain the different aggregated networks from queries. Finally they present the materialization approach. Although this paper is the most advanced research in graph data warehouse, there is still open questions. Indeed, (1) they only consider a graph of vertex of the same type that let all the open questions we introduced unanswered. (2) They consider only local centralized processing and (3) the materialization policy is inspired by legacy data warehouse and does not leverage the distributed graph processing aspects.

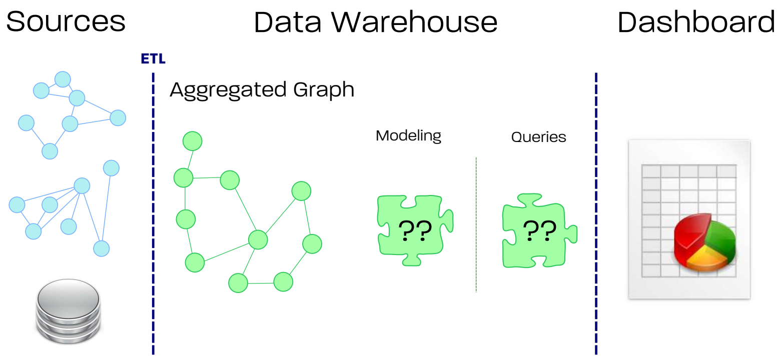


Fig. 5. The graph data warehouse process.

5 Conclusion

The large graph mining is becoming an important requirement coming from the industry and the research community. The graph model leverages the relationship between objects and enables to better structure linked data. In the other hand this model is not well suited for traditional data mining algorithms and processing framework. The mining algorithms need to be re-designed to take into account the structural nature of graphs and to be adapted to distributed programing paradigms to scale. In this paper we presented the *PageRank*, the k-means based and the centrality-based algorithms, we described them and explained how they can be adapted for graph structures. Afterwards, we introduced new emerging graph distributed frameworks and described how the previous mining algorithms can be implemented within their programming models. Finally, we introduced a new field of research, the graph data warehouse, which is deeply linked with large graph mining. This field is still at early stage but dramatically lacks conceptual modeling, unified query model and integration with new distribute programing techniques.

This paper shown that if we consider to analyze significant graphs, distributed programming techniques can be applied to efficiently speed-up the processing. However, the implementation of those techniques and the frameworks is still at early stage, the documentation and the support are clearly lacking. In addition, the implicit distributed programming model can be difficult for porting legacy graph mining algorithms.

References

1. A. Abelló, J. Samos, and F. Saltor. Yam^2: a multidimensional conceptual model extending uml. *Inf. Syst.*, 31(6):541–567, 2006.
2. C. C. Aggarwal and H. Wang, editors. *Managing and Mining Graph Data*, volume 40 of *Advances in Database Systems*. Springer, 2010.
3. Z. E. Akkaoui, E. Zimányi, J.-N. Mazón, and J. Trujillo. A model-driven framework for etl process development. In *DOLAP*, pages 45–52, 2011.
4. A. Avram. Gremlin, a language for working with graphs. Technical report, InfoQ, 2010.
5. D. Bader. Facing the multicore-challenge. chapter Analyzing massive social networks using multicore and multithreaded architectures, pages 1–1. Springer-Verlag, Berlin, Heidelberg, 2010.
6. D. A. Bader and K. Madduri. Snap, small-world network analysis and partitioning: An open-source parallel graph framework for the exploration of large-scale networks. In *IPDPS*, pages 1–12, 2008.
7. A. Balakrishnan, T. L. Magnanti, and R. T. Wong. A Dual-Ascent procedure for Large-Scale uncapacitated network design. *Operations Research*, 37(5):716–740, 1989.
8. A. Bauer, W. Hümmer, and W. Lehner. An alternative relational olap modeling approach. In *DaWaK*, pages 189–198, 2000.
9. R. Bellman. On a routing problem. *Quarterly of Applied Mathematics*, 16:87–90, 1958.
10. A. Bialecki, M. Cafarella, D. Cutting, and O. O'Malley. Hadoop: A framework for running applications on large clusters built of commodity hardware, http://lucene.apache.org/hadoop/, 2005.
11. A. Borodin, G. O. Roberts, J. S. Rosenthal, and P. Tsaparas. Finding authorities and hubs from link structures on the world wide web. In *Proceedings of the 10th international conference on World Wide Web*, WWW '01, pages 415–429, New York, NY, USA, 2001. ACM.
12. R. A. Botafogo, E. Rivlin, and B. Shneiderman. Structural analysis of hypertexts: Identifying hierarchies and useful metrics. *ACM Trans. Inf. Syst.*, 10(2):142–180, 1992.

13. T. Brants, A. C. Popat, P. Xu, F. J. Och, J. Dean, and G. Inc. Large language models in machine translation. In *In EMNLP*, pages 858–867, 2007.
14. S. Brin and L. Page. The anatomy of a large-scale hypertextual web search engine. *Computer Networks*, 30(1-7):107–117, 1998.
15. Y. Bu, B. Howe, M. Balazinska, and M. D. Ernst. Haloop: efficient iterative data processing on large clusters. *Proc. VLDB Endow.*, 3:285–296, September 2010.
16. S. Chaudhuri and U. Dayal. An overview of data warehousing and olap technology. *SIGMOD Record*, 26(1):65–74, 1997.
17. M.-S. Chen, J. Han, and P. S. Yu. Data mining: An overview from a database perspective. *IEEE Trans. Knowl. Data Eng.*, 8(6):866–883, 1996.
18. R. Chen, X. Weng, B. He, and M. Yang. Large graph processing in the cloud. In *Proceedings of the 2010 international conference on Management of data*, SIGMOD '10, pages 1123–1126, New York, NY, USA, 2010. ACM.
19. F. R. K. Chung. A local graph partitioning algorithm using heat kernel pagerank. *Internet Mathematics*, 6(3):315–330, 2009.
20. D. Cohn and H. Chang. Learning to probabilistically identify authoritative documents. In P. Langley, editor, *ICML*, pages 167–174. Morgan Kaufmann, 2000.
21. D. Datta and J. R. Figueira. Graph partitioning by multi-objective real-valued metaheuristics: A comparative study. *Appl. Soft Comput.*, 11(5):3976–3987, 2011.
22. E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1):269–271, Dec. 1959.
23. J. Ekanayake, H. Li, B. Zhang, T. Gunarathne, S.-H. Bae, J. Qiu, and G. Fox. Twister: a runtime for iterative mapreduce. In *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, HPDC '10, pages 810–818, New York, NY, USA, 2010. ACM.
24. U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, editors. *Advances in Knowledge Discovery and Data Mining.* AAAI/MIT Press, 1996.
25. G. Fedak, G. Fox, G. Antoniu, and H. He. Future of mapreduce for scientific computing. In *Proceedings of the second international workshop on MapReduce and its applications*, MapReduce '11, pages 75–76, New York, NY, USA, 2011. ACM.
26. R. W. Floyd. Algorithm 97: Shortest path. *Commun. ACM*, 5:345–, June 1962.
27. L. Freeman. Centrality in social networks conceptual clarification. *Social networks*, 1(3):215–239, 1979.
28. B. Gaujal, N. Navet, and C. Walsh. Shortest-path algorithms for real-time scheduling of fifo tasks with minimal energy use. *ACM Trans. Embed. Comput. Syst.*, 4:907–933, November 2005.
29. M. Girvan and M. E. J. Newman. Community structure in social and biological networks. *Proceedings of the National Academy of Sciences*, 99(12):7821–7826, 2002.
30. R. Gupta and S. K. Malik. Sparql semantics and execution analysis in semantic web using various tools. In *Proceedings of the 2011 International Conference on Communication Systems and Network Technologies*, CSNT '11, pages 278–282, Washington, DC, USA, 2011. IEEE Computer Society.
31. J. Han and M. Kamber. *Data Mining: Concepts and Techniques.* Morgan Kaufmann, 2000.
32. B. Husemann, J. Lechtenbörger, and G. Vossen. Conceptual data warehouse design. In *Proc. of the International Workshop on Design and Management of Data Warehouses (DMDW 2000)*, pages 3–9, 2000.
33. T. Imielinski and H. Mannila. A database perspective on knowledge discovery. *Commun. ACM*, 39(11):58–64, 1996.

34. M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly. Dryad: distributed data-parallel programs from sequential building blocks. In *Proceedings of the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007*, EuroSys '07, pages 59–72, New York, NY, USA, 2007. ACM.
35. A. Jain, M. Murty, and P. Flynn. Data clustering: a review. *ACM computing surveys (CSUR)*, 31(3):264–323, 1999.
36. D. R. Kaeli, L. L. Fong, R. C. Booth, K. C. Imming, and J. P. Weigel. Performance analysis on a cc-numa prototype. *IBM J. Res. Dev.*, 41:205–214, May 1997.
37. U. Kang, C. Tsourakakis, A. Appel, C. Faloutsos, and J. Leskovec. Hadi: Fast diameter estimation and mining in massive graphs with hadoop. *CMU-ML-08-117*, 2008.
38. U. Kang, C. E. Tsourakakis, and C. Faloutsos. Pegasus: A peta-scale graph mining system. In W. Wang, H. Kargupta, S. Ranka, P. S. Yu, and X. Wu, editors, *ICDM*, pages 229–238. IEEE Computer Society, 2009.
39. L. Kaufman and P. J. Rousseeuw. *Finding Groups in Data: An Introduction to Cluster Analysis (Wiley Series in Probability and Statistics).* Wiley-Interscience, Mar. 2005.
40. M. Khosrow-Pour, editor. *Encyclopedia of Information Science and Technology (5 Volumes).* Idea Group, 2005.
41. R. Kimball and M. Ross. *The Data Warehouse Toolkit: The Complete Guide to Dimensional Modeling.* John Wiley & Sons, Inc., New York, NY, USA, 2nd edition, 2002.
42. J. M. Kleinberg. Authoritative sources in a hyperlinked environment. In H. J. Karloff, editor, *SODA*, pages 668–677. ACM/SIAM, 1998.
43. J. M. Kleinberg. Authoritative sources in a hyperlinked environment. *J. ACM*, 46:604–632, September 1999.
44. R. Kumar, J. Novak, and A. Tomkins. Structure and evolution of online social networks. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '06, pages 611–617, New York, NY, USA, 2006. ACM.
45. R. Lämmel. Google's mapreduce programming model revisited. *Sci. Comput. Program.*, 70:1–30, January 2008.
46. J. Leskovec, K. J. Lang, A. Dasgupta, and M. W. Mahoney. Statistical properties of community structure in large social and information networks. In *Proceedings of the 17th international conference on World Wide Web*, WWW '08, pages 695–704, New York, NY, USA, 2008. ACM.
47. C. Liu, F. Guo, and C. Faloutsos. Bbm: bayesian browsing model from petabyte-scale data. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '09, pages 537–546, New York, NY, USA, 2009. ACM.
48. D. H. Lorenz and A. Orda. Qos routing in networks with uncertain parameters. *IEEE/ACM Trans. Netw.*, 6:768–778, December 1998.
49. S. Luján-Mora, J. Trujillo, and I.-Y. Song. A uml profile for multidimensional modeling in data warehouses. *Data Knowl. Eng.*, 59(3):725–769, 2006.
50. J. B. MacQueen. Some methods for classification and analysis of multivariate observations. In L. M. L. Cam and J. Neyman, editors, *Proc. of the fifth Berkeley Symposium on Mathematical Statistics and Probability*, volume 1, pages 281–297. University of California Press, 1967.
51. G. Malewicz, M. H. Austern, A. J. C. Bik, J. C. Dehnert, I. Horn, N. Leiser, and G. Czajkowski. Pregel: a system for large-scale graph processing. In A. K.

Elmagarmid and D. Agrawal, editors, *SIGMOD Conference*, pages 135–146. ACM, 2010.
52. E. Malinowski and E. Zimányi. *Advanced data warehouse design: From conventional to spatial and temporal applications*. Springer-Verlag, 2008.
53. E. Malinowski and E. Zimányi. Multidimensional conceptual modeling. In J. Wang, editor, *Encyclopedia of Data Warehousing and Mining*, pages 293–300. IGI Global, second edition, 2008.
54. M. Marchiori. The quest for correct information on the web: Hyper search engines. *Computer Networks*, 29(8-13):1225–1236, 1997.
55. N. Martínez-Bazan, V. Muntés-Mulero, S. Gómez-Villamor, J. Nin, M.-A. Sánchez-Martínez, and J.-L. Larriba-Pey. Dex: high-performance exploration on large graphs for information retrieval. In *Proceedings of the sixteenth ACM conference on Conference on information and knowledge management*, CIKM '07, pages 573–582, New York, NY, USA, 2007. ACM.
56. F. McSherry. Spectral partitioning of random graphs. In *FOCS*, pages 529–537, 2001.
57. E. N. Mortensen and W. A. Barrett. Interactive segmentation with intelligent scissors. *Graph. Models Image Process.*, 60:349–384, September 1998.
58. C. Nolan. Manipulate and query olap data using adomd and multidimensional expressions. Technical report, Microsoft Research, 1999.
59. L. Page, S. Brin, R. Motwani, and T. Winograd. The pagerank citation ranking: Bringing order to the web. Technical Report 1999-66, Stanford InfoLab, November 1999. Previous number = SIDL-WP-1999-0120.
60. X. Qiu, J. Ekanayake, S. Beason, T. Gunarathne, G. Fox, R. Barga, and D. Gannon. Cloud technologies for bioinformatics applications. In *Proceedings of the 2nd Workshop on Many-Task Computing on Grids and Supercomputers*, MTAGS '09, pages 6:1–6:10, New York, NY, USA, 2009. ACM.
61. F. Radicchi, C. Castellano, F. Cecconi, V. Loreto, and D. Parisi. Defining and identifying communities in networks. *Proceedings of the National Academy of Sciences of the United States of America*, 101(9):2658, 2004.
62. M. J. Rattigan, M. E. Maier, and D. Jensen. Graph clustering with network structure indices. In Z. Ghahramani, editor, *ICML*, volume 227 of *ACM International Conference Proceeding Series*, pages 783–790. ACM, 2007.
63. S. Rizzi. Conceptual modeling solutions for the data warehouse. In *Database Technologies: Concepts, Methodologies, Tools, and Applications*, pages 86–104. 2009.
64. M. A. Rodriguez and P. Neubauer. A path algebra for multi-relational graphs. In *Proceedings of the 2011 IEEE 27th International Conference on Data Engineering Workshops*, ICDEW '11, pages 128–131, Washington, DC, USA, 2011. IEEE Computer Society.
65. C. Sapia, M. Blaschka, G. Höfling, and B. Dinter. Extending the e/r model for the multidimensional paradigm. In *ER Workshops*, pages 105–116, 1998.
66. A. Schätzle, M. Przyjaciel-Zablocki, and G. Lausen. Pigsparql: mapping sparql to pig latin. In *Proceedings of the International Workshop on Semantic Web Information Management*, SWIM '11, pages 4:1–4:8, New York, NY, USA, 2011. ACM.
67. T. Segaran, C. Evans, and J. Taylor. *Programming the Semantic Web - Build Flexible Applications with Graph Data*. O'Reilly, 2009.
68. C. Sommer. *Approximate Shortest Path and Distance Queries in Networks*. PhD thesis, University of Tokyo, 2010.
69. X. Sui, D. Nguyen, M. Burtscher, and K. Pingali. Parallel graph partitioning on multicore architectures. In K. D. Cooper, J. M. Mellor-Crummey, and V. Sarkar,

editors, *LCPC*, volume 6548 of *Lecture Notes in Computer Science*, pages 246–260. Springer, 2010.

70. N. Tryfona, F. Busborg, and J. G. B. Christiansen. Starer: A conceptual model for data warehouse design. In *DOLAP*, pages 3–8, 1999.
71. L. G. Valiant. A bridging model for parallel computation. *Commun. ACM*, 33:103–111, August 1990.
72. S. Wasserman and K. Faust. *Social Network Analysis: Methods and Applications*. Number 8 in Structural analysis in the social sciences. Cambridge University Press, 1st edition, 1994.
73. F. B. Zhan and C. E. Noon. Shortest path algorithms: An evaluation using real road networks. *Transportation Science*, 32:65–73, 1998.
74. P. Zhao, X. Li, D. Xin, and J. Han. Graph cube: on warehousing and olap multidimensional networks. In *Proceedings of the 2011 international conference on Management of data*, SIGMOD '11, pages 853–864, New York, NY, USA, 2011. ACM.
75. X. Zhao, A. Sala, C. Wilson, H. Zheng, and B. Y. Zhao. Orion: shortest path estimation for large social graphs. In *Proceedings of the 3rd conference on Online social networks*, WOSN'10, pages 9–9, Berkeley, CA, USA, 2010. USENIX Association.
76. A. Zhou, W. Qian, D. Tao, and Q. Ma. Disg: A distributed graph repository for web infrastructure (invited paper). In *ISUC*, pages 141–145. IEEE Computer Society, 2008.
77. L. Zhuang, J. Dunagan, D. R. Simon, H. J. Wang, and J. D. Tygar. Characterizing botnets from email spam records. In *Proceedings of the 1st Usenix Workshop on Large-Scale Exploits and Emergent Threats*, pages 2:1–2:9, Berkeley, CA, USA, 2008. USENIX Association.