Front. Comput. Sci., 2016, 10(3): 387–398
DOI 10.1007/s11704-015-4515-1

REVIEW ARTICLE

Big graph search: challenges and techniques

Shuai MA, Jia LI, Chunming HU (✉), Xuelian LIN, Jinpeng HUAI

State Key Laboratory of Software Development Environment, School of Computer Science and Engineering,
Beihang University, Beijing 100191, China

Abstract On one hand, compared with traditional relational and XML models, graphs have more expressive power and are widely used today. On the other hand, various applications of social computing trigger the pressing need of a new search paradigm. In this article, we argue that big graph search is the one filling this gap. We first introduce the application of graph search in various scenarios. We then formalize the graph search problem, and give an analysis of graph search from an evolutionary point of view, followed by the evidences from both the industry and academia. After that, we analyze the difficulties and challenges of big graph search. Finally, we present three classes of techniques towards big graph search: query techniques, data techniques and distributed computing techniques.

Keywords graph search, big data, query techniques, data techniques, distributed computing

1 Introduction

With the rapid development of social computing, Internet and various applications have brought about exponentially growing data. According to the recent report of the UN's international telecommunications union (ITU), Internet users will hit 3 billion globally by the end of 2014[1]; The total number of monthly active Facebook users has reached over 1.3 billion, and the increment of its users from 2012 to 2013 is about 22%[2]. All these indicate the coming of an era of big data. Indeed, "data are becoming the new raw material of business: an economic input almost on a par with capital and labour [1]". How to filter unnecessary data and find the desired information so that one could easily make timely and accurate decisions? This has become one of the most pressing needs in such a big data era.

Compared with traditional relational and XML models, graphs have more expressive power, and play an important role in many applications, such as social networks, biological data analyses, recommender systems, complex object identification and software plagiarism detection. Essentially, this is because the core data involved in these applications can be conveniently represented as graphs. For instance, a social network (e.g., Facebook[3], Twitter[4] and Weibo[5]) has all kinds of social users/activities, which is essentially a graph, whose nodes denote users/activities and edges denote their relationships, such as friendships, respectively.

The wide use of graphs has brought about the emergence of big graph search, i.e., retrieving information from big graphs in a timely and accurate manner, which has drawn more and more attention from both the industry and academia [2–4]. We first give an overview of the application scenarios of graph search.

(1) Social networks and the Web

Nowadays, the rapid development of the Web and social networks has made significant influences on people's social

Received November 13, 2014; accepted January 12, 2015

E-mail: hucm@buaa.edu.cn

[1] http://www.itu.int/en/ITU-D/statistics
[2] http://www.statisticbrain.com/facebook-statistics/
[3] http://www.facebook.com
[4] http://twitter.com
[5] http://weibo.com/

and personal behaviors. Take for instance, Facebook: (a) the total number of its users is very large: there are more than 1.3 billion monthly active users and 0.68 billion mobile users till June 2014; (b) the relations among users and other objects are tight: a user has 130 friends and likes 80 pages on average; (c) there is a large amount of information dissemination on Facebook: more than 4.75 billion pieces of content are shared daily; (d) the site visit of Facebook is quite frequent: 23% of users check Facebook five times or more daily, and a user spends 20 minutes on the site per visit on average[2)].

As mentioned earlier, social networks can be easily represented by graphs, which comes with all kinds of graph search techniques [5–8], including neighbor query and social network compression [9]. Similar to social networks, the Web can be expressed as a big graph as well, whose nodes denote Web pages, and edges indicate hyperlink relationships between Web pages. In fact, the Web site classification and Web mirror detection problems can be treated as the graph classification [10] and graph matching problems [11], respectively.

(2) Recommender systems

Recommendation has found its usage in many applications, such as social matching systems, and graph search is a useful tool for recommendation [12]. Consider the example that a headhunter wants to find a biologist (Bio) to help a group of software engineers (SEs) analyze genetic data [13, 14]. To do this, she uses an expertise recommendation network G, as depicted in Fig. 1, in which nodes denote persons labeled with their expertise, and edges indicate recommendations, e.g., HR_1 recommends Bio_1, and AI_1 recommends DM_1. The biologist Bio needed is specified with a pattern graph Q, also shown in Fig. 1. We could find that Bio has to be recommended by: (a) an HR, an SE and a data mining expert (DM) together, as data mining knowledge is required for the job, (b) the SE is also recommended by the HR, and (c) there is an artificial intelligence expert (AI) who recommends the DM and is recommended by the DM. Based on the pattern graph Q and data graph G, the headhunter could find the suitable biologist in G who meets the requirements, by utilizing graph search techniques developed in [13, 14].

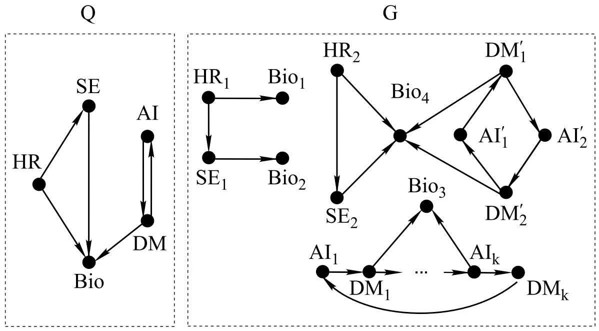

Fig. 1 A recommendation network

(3) Complex object identification

Data quality problem costs U.S. business more than $600 billion a year [15], and data cleaning techniques can help mitigate the losses to a large extent, e.g., it delivers an overall business value of more than "600 million GBP" each year at BT by adopting data cleaning tools [16]. Data cleaning typically contains two central issues: record matching and data repairing [17]. Complex object identification is the most difficult issue in record matching, which is to identify complex objects referring to the same entity in a physical world. One possible solution is to represent complex objects as graphs, and then to identify the same ones by utilizing graph search techniques, such as subgraph isomorphism and graph homomorphism [11, 18].

(4) Software plagiarism detection

With the popularity of open-source software, it gets much easier for a less self-disciplined developer to use (part of) other software without giving proper credits. Traditional plagiarism detection tools are not adequate for finding serious software plagiarism cases. A novel plagiarism detection tool has been developed based on graph search techniques [19]. Firstly, it transforms the source and target programs into program dependence graphs [20]. Secondly, it tests the similarity of the two program dependence graphs with subgraph isomorphism [18]. Finally, if the graph similarity is high enough, it concludes the plagiarism. The rational behind this is that the core control flow of programs, reflected by their program dependence graphs, are hardly to be modified.

(5) Traffic route planning

Graph search is a common practice in transportation networks, due to the wide application of location-based services. Consider an example taken from [21]. Mark is a driver in the U.S. who wants to travel from Irvine to Riverside in California. (a) If Mark wants to reach Riverside by his car in the shortest time, this can be treated as the classical shortest path problem [22], based on which Mark can figure out his best solution from Irvine to Riverside is by traveling along State Route 261, as illustrated by Fig. 2(b) However, if Mark drives a truck carrying with hazardous materials, which may not be allowed to cross over some bridges or railroad crossings, then a pattern graph approach specifying route constraints with regular expressions may be needed to find an optimal transport route [23].

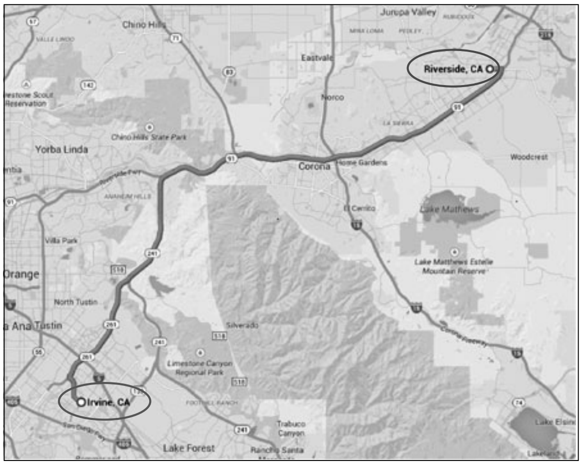

Fig. 2 A route planning example

In addition, graph search techniques have also been adopted in virtual networks [24], pattern recognition [25] and VLSI design [26], among other things.

In the rest of this article, we first give a formal definition of graph search and explain why it is important in Section 2. Then we introduce the challenges of big graph search in Section 3, followed by techniques towards big graph search in Section 4. Finally, we conclude in Section 5.

2 Graph search, why bother?

In this section, we first formalize the concept of graph search. Then we give an analysis of graph search from an evolutionary point of view and point out its urgent need, followed by the evidences from the industry and academia.

2.1 What is graph search

We first formalize the concept of graph search:

Given two graphs G_p, also referred to as the pattern graph and G_d, also referred to as the data graph, graph search is (1) to decide whether G_p "matches" G_d, or (2) to identify the subgraphs of G_d that G_p "matches".

Here graphs consist of nodes and edges, both of which are often attached with labels indicating all kinds of information. Pattern graphs are usually small, e.g., with several or dozens of nodes/edges, while data graphs are often big, e.g., with billions of nodes/edges.

Graph search covers two classes of queries: (1) the first class is boolean queries, i.e., to answer "yes" or "no", and (2) the second one is functional queries, i.e., to identify and return the matching subgraphs. It is obvious that functional queries may need the aid of boolean queries.

Remarks. The above definition of graph search is quite general, as different semantics of "match" lead to different graph search queries [2, 3]. Most, if not all, common graph queries belong to graph search queries, such as node queries (e.g., neighbor query [9]), path queries (e.g., reachability [27] and shortest path [22]) and subgraph queries (e.g., graph homomorphism [11], subgraph isomorphism [18], graph simulation [6] and its extension strong simulation [13]).

2.2 An evolutionary point of view

A serious question arises naturally: why do we need another search paradigm — graph search? We next answer this question from an evolutionary point of view.

Consider the evolution roadmap of information search shown in Fig. 3. The emphasis of information search has undergone a serious shift, i.e., from file systems, to database systems, to the World Wide Web, and to the most recent social networks:

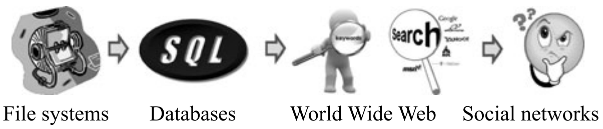

Fig. 3 The evolution of information search

- *File systems*. Since the 1960s, computers have been equipped with modern operating systems [28]. The file system in an operating system is an abstraction to store and organize a set of computer files, and it usually supports users to look for specific files, i.e., simple searching functionalities.
- *Database systems*. In the mid-1960s, database systems began being applied in business, and, subsequently, relational databases played a dominant role. Since the late 1970s, the invention of structured query language (SQL) has significantly promoted the use of databases [29].
- *The Web*. Since the 1990s, search engines, such as Google, Bing and Yahoo!, have been widely used due to the blossom of the World Wide Web. These search engines unanimously adopt the simple but very useful approach — keyword search, which provides people with a convenient and easy way to search specific information on the Web.
- *Social networks*. From the end of last century, with the rising of Web 2.0, social networks have made significant influences on the society. However, a dominant search paradigm seems missing in such an era of social

computing and big data.

As the above analysis shows, an important IT invention, e.g., file systems, database systems and the Web, usually triggers the emergence of a novel search paradigm. We are essentially in a situation to look for one for social computing and social networks, and we believe that *graph search is the one filling the gap*. The "graph search"[6] and "knowledge graph"[7] released by Facebook and Google, respectively, shed light on this. However, another question arises: why could not we simply use SQL or keyword based search?

(1) Graph search vs. SQL search

SQL search is a very strong supporting tool for searching information from relational database systems. However, it is not appropriate for searching information from graphs even though graphs could be stored using relations, due to its disability and inconvenience for answering recursive queries such as graph reachability and shortest paths [30]. Indeed, for simple graph queries that SQL search would do, graph search could do even better. We next illustrate this with an example taken from [31], a simple searching case of "finding the names of all of Alberto Pepe's friends in a social network".

Case 1: Social networks are stored using relations

There are two relations: person(identifier, name) for storing a person's unified identifier and its name, and friend(person_a, person_b) for storing the friendship of persons with identifiers person_a and person_b. In addition, two B^{+}*–tree* indexes are built on each column of the person relation: the person.identifier and person.name indexes, and one index is built on the person_a column of the friend relation: the friend.person_a index. We assume that there are in total n persons and m friendships. The relational representation is presented in Fig. 4.

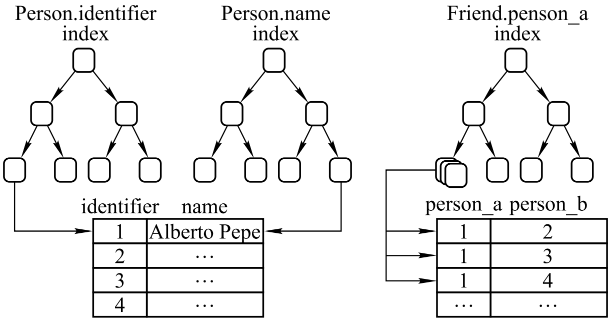

Fig. 4 Relational representation

To get the names of Alberto Pepe's friends, three steps are necessary, as shown in the following.

(a) Find the unique identifer of "Alberto Pepe" from relation person, which takes $O(\log_2 n)$ time using the person.name index.

(b) Find all the k identifiers of the friends of "Alberto Pepe" from relation friend with the identifer found in (a), which takes $O(\log_2 n+k)$ time using the friend.person_a index.

(c) Find the k friends' names from relation person with the k identifiers found in (b), which takes $O(k \log_2 n)$ time using the person.identifier index.

Case 2: Social networks are stored using naive graphs

The person and friendship information can be stored as a graph as shown in Fig. 5. Each person can be represented as a node labeled with the person's name and unified identifier, and the friendship between two persons can be represented as an edge between the two corresponding nodes. A B^{+}*–tree* index is built on the graph, vertex.name index, to quickly locate the position of a node in the graph with a person's name.

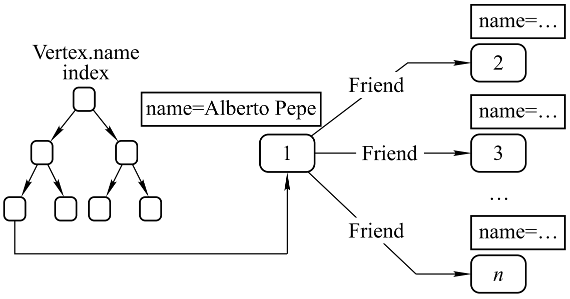

Fig. 5 Graph representation

To get the names of Alberto Pepe's friends, two steps are needed, as shown below.

(a) Identify the node with name "Alberto Pepe", which takes $O(\log_2 n)$ time using the vertex.name index.

(b) Find the k friend nodes of the node found in (a) by traversing its adjacent neighboring nodes and get the friend names directly in the k node labels, which takes $O(k + y)$ time such that $k + y$ is the total number of the neighboring nodes.

It is obvious that the searching speed is improved from $O((k + 2) \log_2 n)$ to $O(\log_2 n)$ when using the graph representation, instead of the relational representation. The improvement is crucial when n is really large, e.g., when there are billions of users. Of course, one could add redundant infor-

[6] https://www.facebook.com/about/graphsearch

[7] http://www.google.com/insidesearch/features/search/knowledge.html

mation to speed up its efficiency, which results in extra space cost in turn. Hence, for big graph search, the graph search approach is much superior to the SQL search approach.

(2) Graph search vs. keyword search

The traditional keyword based searching approach is mainly for retrieving information from the Web, which is not appropriate for searching information from social networks. The information on the Web is usually isolated and *object–object weak tied* from each other, and mainly about "historical and existing" information, i.e., what happened and happening. Social computing generally takes the social factors into consideration, such as the social structure, organization and activity, which makes *relations* a dominant role in social search. Besides, social data are usually *person–person strong related* or *person–object strong related*. This makes the *future and relation* information particularly important for social search. Under these circumstances, the keyword based searching approaches cannot meet the requirements raised by social computing and social networks nowadays.

Hence we argue that graph search is a new searching paradigm for social computing in the big data era. Indeed, Facebook has provided a new searching technique named "Graph Search", which allows users to search for information using simple natural language sentences, e.g., "Restaurants in New York that my friends like", "Photos taken in Hawaii of my friends" and "National parks where my friends have been to". Besides, the development of social networks has also promoted the urgent need of a new search engine in turn.

2.3 Joint efforts of the industry and academia

Recently, we have conducted a survey on the number of papers on graphs published in the top three influential database system conferences (SIGMOD, VLDB and ICDE) ever since 2000. The result is shown in Fig. 6, from which we have found that: (a) from around 2000 (the emergence of Web 2.0), researchers began to focus on the study of graphs, (b) the number of papers on graphs has been increasing continuously since then, (c) from 2008, graphs have been a hot topic in the field of database research, and (d) there is a burst of the number of papers on graphs in 2014.

Many well-known research institutions and companies have been concentrating on the research and applications of graphs. For example, Microsoft's Trinity project[8] and "Horton - Querying Large Distributed Graphs" project[9] for data center; large-scale graph processing system Pregel [32] of Google; "Knowledge Acquisition and Management" project[10] of Yahoo!; Neo4j's open-source graph database[11]; "Graph Search" of Facebook[6]; and the research teams from academia such as the University of California Santa Barbara, University of Edinburgh, University of New South Wales, Chinese University of Hong Kong, and Beihang University.

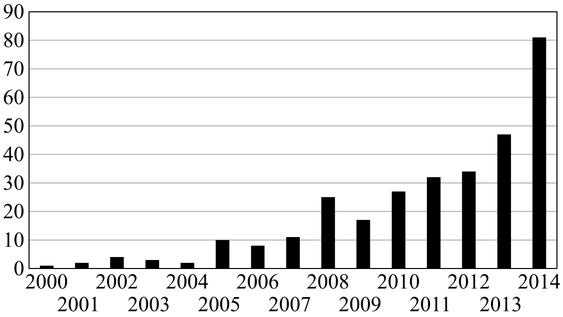

Fig. 6 Statistics of papers (published in SIGMOD, VLDB and ICDE) on graphs

The joint interests and efforts from both the industry and academia provide more evidences on the power and importance of graph search.

3 Challenges of big graph search

In this section, we first introduce the FAE rule that is important for a search engine, and we then point out its difficulties and challenges for big graph search.

3.1 The FAE rule

The FAE rule says that the quality of search engines involves with three key factors: *friendliness*, *accuracy* and *efficiency*, as illustrated in Fig. 7, and that a good search engine must provide the users with a friendly query interface and highly accurate answers in a fast way.

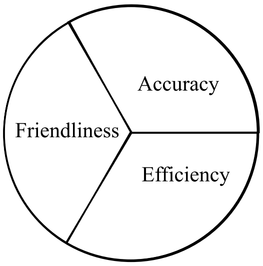

Fig. 7 The FAE rule

[8] http://research.microsoft.com/en-us/projects/trinity

[9] http://research.microsoft.com/en-us/projects/ldg

[10] http://research.yahoo.com/project/

[11] http://neo4j.org

(1) Friendliness

It is necessary for a search engine to provide the users with a friendly query interface such that the users could conveniently specify their searching conditions with small efforts.

Generally speaking, the keyword search on the Web only requires users to enter several keywords, which is very user-friendly. However, it cannot allow users to specify complex search conditions like graphs (such as relationships among keywords), and it only returns the Web hyperlinks that might contain answers to users. Hence, this simpleness also brings the gap between what the users want and what the users get. In contrast, the results of graph search are much more accurate as it allows users to further specify structural constraints by designing various pattern graphs. However, it is definitely inconvenient for users to enter pattern graphs as inputs even for small pattern graphs, as it is hard for non-professional users who are not familiar with the complex data graphs to specify precise pattern graphs.

People are already making an effort for designing friendly graph search interfaces. The technique developed by Facebook allows users to specify pattern graphs with simple natural language sentences, as we mentioned earlier. And Yang et al. [33] have recently proposed a novel graph search system enabling schemaless and structureless graph querying, which (a) provides a user-friendly interface where users can give rough descriptive pattern graphs as queries, and (b) supports various kinds of transformations such as synonym, abbreviation, and ontology. However, a completely friendly interface meeting the requirements of practical applications is still on its road for big graph search.

(2) Accuracy

It is necessary for a search engine to provide the users with accurate answers.

When a user submits a query to a search engine, which represents the user's searching goal, the search engine analyzes the user's input and tries to understand what the user wants. Hence, to reach high searching accuracy, it is indispensable to understand the users' real intents for search engines. However, it is pretty common that there is a gap between what a user wants and what she/he gets back from a search engine. This is because it is a very challenging task to understand and specify the users' intents in a way such that a machine could easily understand. For example, when a user submits "*apple*" to a search engine, it is hard to distinguish the fruit apple from the products of Apple Inc..

Common approaches [34,35] focus on query classification. Given a query, these approaches try to classify the query to some predefined classes. Recently, some researchers take into account of the difference of individuals and attempt to analyze the intents of users by incorporating their search behaviors and preferences [36, 37].

Knowledge also plays an important role to understand the user intent and to improve the searching accuracy. For example, knowledge graph makes Google search engine more intelligent to understand the searching intents of users. When having the keyword "apple" into Google search engine, it will provide two extra panels in addition to a list of Web hyperlinks, one for Apple Inc. and the other for the apple fruit. Then users can click one to enlarge and get detailed information based on their intents, which allows users to get more relevant results without having to visit other Web sites to judge whether the information are relevant by themselves. This is because Google now is able to understand the difference among these entities, and the nuance in their meanings, with the aid of Knowledge Graph[12].

(3) Efficiency

How to search information in a fast way is a key for the success of a search engine. It is also a fundamental problem in database and information retrieval areas, especially when we are dealing with big graphs today. We will introduce several searching techniques for big graphs in detail in the coming Section 4.

3.2 The challenges

The expressiveness of graphs naturally comes with more difficulties, and the emerging social applications raise more challenges to search and manage big graphs.

According to statistics, for Facebook, there are over 1.3 billion monthly active users; for every 20 minutes, there are 1 million links shared, 2 million friend requests generated, and 3 million messages sent[2]; similarly for Twitter, there are over 0.6 billion users; every second there are 9 100 tweets happened; and people query twitter search engine 2.1 billion times every day[13].

These statistics indicate the following. (a) Graph data have reached hundred millions orders of magnitude [38]; (b) Graph data are updated all the time, and the update amount daily reaches hundred thousands orders of magnitude [39]; (c) Similar to traditional relational data [40, 41], graph data have the data uncertainty problem due to the external reason caused by data sampling and data missing and the internal

[12] http://googleblog.blogspot.com/2012/05/introducing-knowledge-graph -things-not.html
[13] http://www.statisticbrain.com/twitter-statistics

reason caused by the dynamic changes in graph data; And, even worse, (d) graph data are much more complex than relational and XML data. In summary, graph data have four key features: *big*, *dynamic*, *uncertain* and *complex* [3]. The first feature requires that graph search needs to strike a balance between its time and space cost. When the graph is too large to be processed on single machines, it is also necessary to design efficient and effective distributed algorithms. The second feature requires that graph search should take dynamic changes and temporal factors into consideration. The last two features require that graph search should design reasonable models to capture uncertainties in graph data, and highly efficient algorithms to answer graph search queries on uncertain graphs.

These together make it an extremely challenging task to develop a big graph search engine with a friendly query interface, accurate answers and high efficiency.

4 Techniques towards big graph search

A fundamental issue in the big data era is the efficiency. In this section, we present three classes of techniques for big graph search: query techniques, data techniques and distributed computing techniques.

4.1 Query techniques

We first introduce two query techniques: query approximation and incremental computation.

(1) Query approximation

The core idea of query approximation is to transform a class of queries Q with higher computational complexity into another class of queries Q' with lower computational complexity and satisfiable approximate answers, as depicted in Fig. 8 in which Q, Q' and D denote the original query, approximate query and data, respectively. The major challenge comes from the need of a balance between the query efficiency and answer accuracy.

$$\begin{tikzcd} Q(D) \arrow[r, dashed] \arrow[r, bend left, "\text{Approximation}"] & Q'(D) \end{tikzcd}$$

Fig. 8 Query approximation

We next explain the query approximation technique using *strong simulation*, a new graph pattern matching model proposed in [13, 14]. Graph pattern matching is to find all matched subgraphs in a data graph for a given pattern graph, and it is often defined in terms of *subgraph isomorphism*. The goodness of subgraph isomorphism is that all matched subgraphs are exactly the same as the pattern graph, i.e., completely preserving the topology structure between the pattern graph and data graph.

Subgraph isomorphism is, however, NP-complete [18], and may return exponential many matched subgraphs. Recent evidences have shown that subgraph isomorphism is too restrictive to find sensible matches in certain scenarios [6]. These hinder the usability of graph pattern matching in emerging applications.

To lower the high complexity of subgraph isomorphism, various extensions of graph simulation [42] have been considered instead in [6, 27]. These extensions allow graph pattern matching to be conducted in cubic-time. However, they fall short of capturing the topology of data graphs, i.e., graphs may have a structure drastically different from pattern graphs they match, and the matches found are often too large to analyze.

To rectify these problems, strong simulation, a revision of graph simulation, was proposed for graph pattern matching, such that strong simulation (a) preserves the topology of pattern graphs and finds a bounded number of matches, (b) retains the same complexity as earlier extensions of graph simulation [6, 27], by providing a cubic-time algorithm for computing strong simulation, and (c) has the locality property that allows us to develop an effective distributed algorithm to conduct graph pattern matching on distributed graphs [13, 14].

(2) Incremental computation

When there are data updates, query answers typically need to be re-computed to reflect the changes. In practice, big data graphs are frequently modified, as we pointed out in Section 2, and it is too costly to recompute matches from scratch every time when the data graphs are updated. Incremental computation is a technique that attempts to reduce time by reusing previous computing efforts and only computing those answers that "depend on" the changed data, and it is depicted in Fig. 9, in which Q, D and Δ denote the query, original data and its updates, respectively.

$$\begin{tikzcd} Q(D+\Delta) \arrow[r, dashed] \arrow[r, bend left] \arrow[rr, bend left, "\text{Ineremental}"] & Q(D) \; + & Q(\Delta) \end{tikzcd}$$

Fig. 9 Incremental computation

It is worth mentioning that incremental algorithms have been developed for various applications (see [43] for a survey). Thomas W. Reps has done pioneering work on the study of incremental computation [43, 44], and he observed that the complexity of incremental algorithms was more accurately characterized in terms of the size of the area affected by the updates, rather than the size of the entire input [44].

Next let's take the indexing of Google search as an example. It is known that the Web documents are crawled and stored in a large repository, and are pre-indexed to speed up the search efficiency and improve the user experiences. The indexing process incurs a heavy workload, and Google initially adopted some batch-processing approaches such as MapReduce [45] to improve the efficiency, which is not satisfactory when facing with constant changes. Google later on developed Percolator [46], a system incrementally processing updates on large data sets. That is, Google has converted its batch-based indexing system into an incremental indexing system. It was reported that compared with MapReduce, Percolator (a) reduced the average document processing latency by a factor of 100, and (b) reduced the average age of resulting documents of Google search by 50% when processing the same amount of documents per day [46].

4.2 Data techniques

One key feature of big data graphs is the large volume, and, hence, the space complexity [47] of graph search starts raising more troubles. Here we introduce five techniques to boost the search efficiency from the data point of view: data approximation, data sampling, data partitioning, data compression and data indexing.

(1) Data approximation

The core idea of data approximation is that given a class of queries Q and a data set D, it transforms D into a smaller data set D' such that Q on D' returns a satisfiable approximate answer in a more efficient way, as depicted in Fig. 10. Similar to query approximation, the major challenge of data approximation comes from the need of a balance between the query efficiency and answer accuracy.

$$\begin{tikzcd}
Q(D) \arrow[r, dashed] \arrow[r, bend left, "\text{Approximation}"] & Q(D')
\end{tikzcd}$$

Fig. 10 Data approximation

We have adopted the idea in the process of dealing with large graphs in the study of anomaly detection in graph streams, when dealing with the matrix representation of a social graph, and we have both theoretically and experimentally shown that simplifying the matrix by replacing a part of small entry values with zero has few affects on the computation of eigenvectors [48].

(2) Data sampling

Sampling is concerned with the selection of a subset of data from a large data set. Instead of dealing with the entire data set D for a query Q, the data sampling technique reduces the size of the data set D by sampling, with a permission of loss of accuracy to some extent in the query result [49]. In a sampling process, it must be ensured that the sampled data Δ obtained must reflect the characteristics and information of the original data D, as depicted in Fig. 11.

$$\begin{tikzcd}
Q(D) \arrow[r, dashed] \arrow[r, bend left, "\text{Sampling}"] & Q(\Delta)
\end{tikzcd}$$

Fig. 11 Data sampling

It is worth mentioning that Michael I. Jordan and his colleagues have proposed a new sampling approach –bootstrap– to dealing with big data [50, 51].

(3) Data partitioning

Data partitioning is an effective method to execute queries on large-scale data sets in a divide-and-conquer way. It partitions a data set D into a set of *relatively small* data sets D_1, $\cdots$, D_n such that $D = D_1 \cup \cdots \cup D_n$. Ideally, the final query answer is assembled using the n answers on the set of small data sets, and the analysis speed can be improved significantly. The entire process is depicted in Fig. 12.

$$\begin{tikzcd}
Q(D) \arrow[r, dashed] \arrow[r, bend left, "\text{Partitioning}"] \arrow[rrr, bend left] & Q(D_1) & + \ldots + & Q(D_n)
\end{tikzcd}$$

Fig. 12 Data partitioning

It is worth mentioning that graph partitioning has been extensively studied since the 1970's [52–54], and has been successfully used in various applications, e.g., circuit placement, parallel computing and scientific simulation [54]. The graph partitioning problem is in general a hard problem and is often NP-complete [53].

(4) Data compression

The principle of data compression is that compressing by removing redundancies remains the capability to answer the same question. There are many known data compression methods that are suitable for different types of data, and produce different answers, but they are all based on the principle, namely compressing data by removing redundancies from the original data (see [55] for a complete reference). The benefits of data compression lie in that it provides more possibilities to work in main memory and potentials to work efficiently.

Different from data sampling, data compression generates a small data set D' from the original data set D by removing redundancies and preserving the information only relevant to queries, as depicted in Fig. 13. In addition, there are usually no restrictions on the formats of the compressed data, while data sampling normally keeps the original data formats. There is a whole bunch of work on (lossy or lossless) graph compression [9, 56–58]. As [59–61] show, some graph algorithms can be speeded up by operating on compressed graphs directly, which can be treated as query oriented compression, and needs to invest more efforts to study.

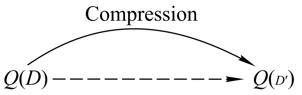

Fig. 13 Data compression

(5) Data indexing

An index is a data structure that improves the speed of queries by reducing search space, at the cost of update maintenance and extra storage. Indexes are commonly used for querying relational databases [29] and information retrieval of search engines [62].

When data graphs are relatively large, graph indexing technique can quickly prune data graphs that obviously mismatch the pattern graph [63]. There already exist indexing methods for (various kinds of) graph pattern matching [49]. There are mainly three metrics for measuring whether an established index is appropriate: the space cost, building time and query time. The smaller the space of an index is, the less additional storage burden incurred. The building time represents the time cost of creating the index, and the query time indicates the time cost for the query process. When data graphs are changed over time, the index refresh speed reflects its adaptiveness to dynamic changes.

4.3 Beyond query and data techniques

We finally introduce the distributed computing technique, as an example that utilizes the above query and data techniques and beyond.

Distributed computing refers to the use of distributed systems to solve problems such that a problem is divided into many tasks, each of which is computed on one or more machines, and which communicate with each other by message passing [64, 65]. Distributed computing typically needs to partition a data set D into *relatively small* data sets $D_1, \ldots, D_n$, and distributes them on multiple computing machines, as depicted in Fig. 14.

It is known that real-life graphs are typically way too large, e.g., the Web graph of Yahoo! has about 14 billion nodes, and there are over 1.3 billion users on Facebook. Hence, it is not practical to handle large graphs on single machines. Moreover, real-life graphs are naturally distributed, e.g., Google, Yahoo! and Facebook have large-scale distributed data centers. This says that distributed computing is inevitable facing with big graphs.

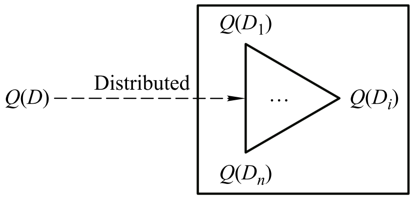

Fig. 14 Distributed computing

We have developed a computation model for a large class of distributed algorithms for graph simulation [66]. The model consists of a cluster of identical machines, in which one acts as a coordinator. Each machine can directly send an arbitrary number of messages to another, and all machines co-work with each other by local computations and message-passing. Further, we also identify three complexity measures on the performance of distributed algorithms related to the computation model above: (a) visit times, which is the maximum visiting times of a machine, indicates the complexity of interactions; (b) makespan, which is the evaluation of the total computation time, is a measure of efficiency; (c) data shipment, which is the size of the total messages shipped among distinct machines during the computation, indicates the network bandwidth consumption. However, these three measures are typically controversial with each other, and how to achieve a balanced strategy is a great challenge for designing distributed algorithms.

Recently, many distributed graph processing systems have been developed, which basically fall into two categories: one makes use of MapReduce [45] or Spark [67] to speed-up big graph processing [68–70], and the other uses different distributed computing models, such as Pregel [32], GraphLab [71] and PowerGraph [72].

Remarks There exists no single technique that could fit all for big graph search. That is, it is often necessary to combine different techniques to obtain satisfiable solutions. We also encourage interested readers to read a very recent article [73] for discussions on the theory and techniques of big data, a complement of this article.

5 Conclusions

In this article we have investigated big graph search, a novel promising search paradigm for social computing in the big data era. First, we have analyzed the need of big graph search with various applications, industrial and academic developments, and the evolution history of information searching paradigms. Second, we have pointed out the challenges and opportunities of big graph search. Finally, we have introduced three types of techniques towards big graph search: query techniques, data techniques and distributed computing techniques.

Being a new paradigm for social computing, big graph search has received extensive attentions. However, there is obviously a long way to go for a big graph search engine that meets various needs in practice.

Acknowledgements This work was supported in part by 973 program (2014CB340300), National Natural Science Foundation of China (Grant No. 61322207) and the Fundamental Research Funds for the Central Universities.

References

1. Cukier K. Data, data everywhere: a special report on managing information. Economist Newspaper, 2010
2. Ma S, Li J, Liu X, Huai J. Graph search: a new searching approach to the social computing era. Communications of CCF, 2012, 8(11): 26–31
3. Ma S, Cao Y, Wo T, Huai J. Social networks and graph matching. Communications of CCF, 2012, 8(4): 20–24
4. Ma S, Li J, Liu X, Huai J. Graph search in the big data era. Information and Communications Technologies, 2013, 6: 44–51
5. Tian Y, Patel J M. Tale: A tool for approximate large graph matching. In: Proceedings of the 24th IEEE International Conference on Data Engineering. 2008, 963–972
6. Fan W, Li J, Ma S, Tang N, Wu Y, Wu Y. Graph pattern matching: from intractable to polynomial time. Proceedings of the VLDB Endowment, 2010, 3(1): 264–275
7. Barcelo P, Hurtado C A, Libkin L, Wood P T. Expressive languages for path queries over graph-structured data. In: Proceedings of the 29th ACM Symposium on Principles of Database Systems. 2010, 3–14
8. Feng K, Cong G, Bhowmick S S, Ma S. In search of influential event organizers in online social networks. In: Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data. 2014, 63–74
9. Maserrat H, Pei J. Neighbor query friendly compression of social networks. In: Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. 2010, 533–542
10. Schenker A, Last M, Bunke H, Kandel A. Classification of web documents using graph matching. International Journal of Pattern Recognition and Artificial Intelligence, 2004, 18(3): 475–496
11. Fan W, Li J, Ma S, Wang H, Wu Y. Graph homomorphism revisited for graph matching. Proceedings of the VLDB Endowment, 2010, 3(1): 1161–1172
12. Terveen L G, McDonald D W. Social matching: a framework and research agenda. ACM Transactions on Computer-Human Interaction, 2005, 12(3): 401–434
13. Ma S, Cao Y, Fan W, Huai J, Wo T. Capturing topology in graph pattern matching. Proceedings of the VLDB Endowment, 2011, 5(4): 310–321
14. Ma S, Cao Y, Fan W, Huai J, Wo T. Strong simulation: capturing topology in graph pattern matching. ACM Transactions on Database Systems, 2014, 39(1)
15. Eckerson W. Data quality and the bottom line: achieving business success through a commitment to high quality data. TDWI Report. 2002
16. Otto B, Weber K. From health checks to the seven sisters: the data quality journey at bt. Report: BT TR-BE HSG/CC CDQ/8. 2009
17. Fan W, Li J, Ma S, Tang N, Yu W. Interaction between record matching and data repairing. In: Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data. 2011, 469–480
18. Ullmann J R. An algorithm for subgraph isomorphism. Journal of the ACM, 1976, 23(1): 31–42
19. Liu C, Chen C, Han J, Yu P S. Gplag: detection of software plagiarism by program dependence graph analysis. In: Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. 2006, 872–881
20. Ferrante J, Ottenstein K J, Warren J D. The program dependence graph and its use in optimization. ACM Transactions on Programming Languages and Systems, 1987, 9(3): 319–349
21. Rice M N, Tsotras V J. Graph indexing of road networks for shortest path queries with label restrictions. Proceedings of the VLDB Endowment, 2010, 4(2): 69–80
22. Cormen T H, Leiserson C E, Rivest R L, Stein C. Introduction to Algorithms. Cambridge: The MIT Press, 2001
23. Chen Z, Shen H T, Zhou X, Yu J X. Monitoring path nearest neighbor in road networks. In: Proceedings of the 2009 ACM SIGMOD International Conference on Management of Data. 2009, 591–602
24. Chowdhury N M M K, Rahman M R, Boutaba R. Virtual network embedding with coordinated node and link mapping. In: Proceedings of the 28th IEEE Conference on Computer Communications. 2009, 783–791
25. Conte D, Foggia P, Sansone C, Vento M. Thirty years of graph matching in pattern recognition. International Journal of Pattern Recognition and Artificial, 2004, 18(3): 265–298
26. Karypis G, Aggarwal R, Kumar V, Shekhar S. Multilevel hypergraph partitioning: applications in vlsi domain. IEEE Transactions on Very Large Scale Integration Systems, 1999, 7(1): 69–79
27. Fan W, Li J, Ma S, Tang N, Wu Y. Adding regular expressions to graph reachability and pattern queries. In: Proceedings of the 27th IEEE Conference on Data Engineering. 2011, 39–50
28. Hansen P B, ed. Classic Operating Systems. New York: Springer, 2001
29. Ramakrishnan R, Gehrke J. Database Management Systems. New York: McGraw-Hill Higher Education, 2000
30. Abiteboul S, Hull R, Vianu V. Foundations of Databases. Addison-Wesley, 1995
31. Sakr S, Pardede E, eds. Graph Data Management: Techniques and Applications. IGI Global, 2011

32. Malewicz G, Austern M H, Bik A J C, Dehnert J C, Horn I, Leiser N, Czajkowski G. Pregel: a system for large-scale graph processing. In: Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data. 2010, 135–146

33. Yang S, Wu Y, Sun H, Yan X. Schemaless and structureless graph querying. Proceedings of the VLDB Endowment, 2014, 7(7): 565–576

34. Beitzel S M, Jensen E C, Frieder O, Lewis D D, Chowdhury A, Kolcz A. Improving automatic query classification via semi-supervised learning. In: Proceedings of the 5th IEEE International Conference on Data Mining. 2005, 42–49

35. Shen D, Sun J T, Yang Q, Chen Z. Building bridges for web query classification. In: Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval. 2006, 131–138

36. Xing Q, Liu Y, Nie J Y, Zhang M, Ma S, Zhang K. Incorporating user preferences into click models. In: Proceedings of the 22nd ACM International Conference on Information and Knowledge Management. 2013, 1301–1310

37. Hu B, Zhang Y, Chen W, Wang G, Yang Q. Characterizing search intent diversity into click models. In: Proceedings of the 20th International Conference on World Wide Web. 2011, 17–26

38. Maria G, Symeon P, Athena V. Massive graph management for the Web and Web 2.0. New Directions in Web Data Management 1. Springer, 2011, 19–58

39. Newman M, Barabási A L, Watts D J. The Structure and Dynamics of Networks. Princeton: Princeton University Press, 2006

40. Rahm E, Do H H. Data cleaning: problems and current approaches. IEEE Data Engineering Bulletin, 2000, 23(4): 3–13

41. Fan W, Li J, Ma S, Tang N, Yu W. Towards certain fixes with editing rules and master data. The International Journal on Very Large Data Bases, 2012, 21(2): 213–238

42. Henzinger M R, Henzinger T A, Kopke P W. Computing simulations on finite and infinite graphs. In: Proceedings of the 36th Annual Symposium on Foundations of Computer Science. 1995, 453–462

43. Ramalingam G, Reps T W. A categorized bibliography on incremental computation. In: Proceedings of the 20th Symposium on Principles of Programming Languages. 1993, 502–510

44. Ramalingam G, Reps T W. On the computational complexity of dynamic graph problems. Theoretical Computer Science, 1996, 158(1): 233–277

45. Dean J, Ghemawat S. Mapreduce: simplified data processing on large clusters. In: Proceedings of the 6th USENIX Conference on Operating System Design and Implementation. 2004, 137–149

46. Peng D, Dabek F. Large-scale incremental processing using distributed transactions and notifications. In: Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation. 2010, 1–15

47. Papadimitriou C H. Computational Complexity. Addison-Wesley, 1994

48. Yu W, Aggarwal C C, Ma S, Wang H. On anomalous hotspot discovery in graph streams. In: Proceedings of the 13th IEEE International Conference on Data Mining. 2013, 1271–1276

49. Aggarwal C C, Wang H. Managing and Mining Graph Data. New York: Springer, 2010

50. Jordan M I. Divide-and-conquer and statistical inference for big data. In: Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. 2012, 4

51. Kleiner A, Talwalkar A, Sarkar P, Jordan M I. The big data bootstrap. In: Proceedings of the 29th International Conference on Machine Learning. 2012, 1759–1766

52. Kernighan B W, Lin S. An efficient heuristic procedure for partitioning graphs. Bell System Technical Journal, 1970, 49(2): 291–307

53. Karypis G, Kumar V. A fast and high quality multilevel scheme for partitioning irregular graphs. SIAM Journal on Scientific Computing, 1998, 20(1): 359–392

54. Yang S, Yan X, Zong B, Khan A. Towards effective partition management for large graphs. In: Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data. 2012, 517–528

55. Salomon D. Data compression: The Complete Reference. 4th ed. New York: Springer, 2007

56. Buehrer G, Chellapilla K. A scalable pattern mining approach to Web graph compression with communities. In: Proceedings of the 2008 International Conference on Web Search and Data Mining. 2008, 95–106

57. Adler M, Mitzenmacher M. Towards compressing Web graphs. In: Proceedings of Data Compression Conference. 2001, 203–212

58. Boldi P, Vigna S. The WebGraph framework I: compression techniques. In: Proceedings of the 13th International Conference on World Wide Web. 2004, 595–602

59. Feder T, Motwani R. Clique partitions, graph compression and speeding-up algorithms. Journal of Computer and System Sciences, 1995, 51(2): 261–272

60. Karande C, Chellapilla K, Andersen R. Speeding up algorithms on compressed Web graphs. In: Proceedings of the 2009 International Conference on Web Search and Data Mining. 2009, 272–281

61. Fan W, Li J, Wang X, Wu Y. Query preserving graph compression. In: Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data. 2012, 157–168

62. Baeza-Yates R A, Ribeiro-Neto B A. Modern Information Retrieval: the concepts and technology behind search. 2nd ed. Harlow: Pearson Education Ltd., 2011

63. Klein K, Kriege N, Mutzel P. CT-Index: Fingerprint-based graph indexing combining cycles and trees. In: Proceedings of the 27th IEEE International Conference on Data Engineering. 2011, 1115–1126

64. Lynch N A. Distributed Algorithms. San Francisco: Morgan Kaufmann, 1996

65. Peleg D. Distributed Computing: A Locality-Sensitive Approach. SIAM, 2000

66. Ma S, Cao Y, Huai J, Wo T. Distributed graph pattern matching. In: Proceedings of the 21st International Conference on World Wide Web. 2012, 949–958

67. Zaharia M, Chowdhury M, Das T, Dave A, Ma J, McCauly M, Franklin M J, Shenker S, Stoica I. Resilient distributed datasets: a fault-tolerant abstraction for in-memory cluster computing. In: Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation. 2012, 15–28

68. Gao J, Zhou J, Zhou C, Yu J X. Glog: A high level graph analysis system using mapreduce. In: Proceedings of the 30th IEEE International Conference on Data Engineering. 2014, 544–555

69. Qin L, Yu J X, Chang L, Cheng H, Zhang C, Lin X. Scalable big graph

processing in mapreduce. In: Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data. 2014, 827–838

70. Xin R S, Gonzalez J E, Franklin M J, Stoica I. Graphx: a resilient distributed graph system on spark. In: Proceeding of the 1st International Workshop on Graph Data Management Experiences and Systems. 2013

71. Low Y, Gonzalez J, Kyrola A, Bickson D, Guestrin C, Hellerstein J M. Distributed graphlab: a framework for machine learning in the cloud. Proceedings of the VLDB Endowment, 2012, 5(8): 716–727

72. Gonzalez J E, Low Y, Gu H, Bickson D, Guestrin C. Powergraph: distributed graph-parallel computation on natural graphs. In: Proceedings of the 10th USENIX Conference on Operating Systems Design and Implementation. 2012, 17–30

73. Fan W, Huai J. Querying big data: bridging theory and practice. Journal of Computer Science and Technology, 2014, 29(5): 849–869

Shuai Ma is a professor in the School of Computer Science and Engineering, Beihang University, China. He obtained his two PhDs from University of Edinburgh, UK in 2010, and from Peking University, China in 2004. He was a postdoctoral research fellow in the database group, University of Edinburgh, and a summer intern at Bell labs, Murray Hill, USA in the summer of 2008. His research interests include database theory and systems, social data analysis, and data intensive computing. He is an Awardee of the NSFC Excellent Young Scholars Program in 2013. Besides, he is a recipient of the best paper award for VLDB 2010, the Visiting Young Faculty Program of MRSA in 2012, and the best challenge paper award for WISE 2013.

Jia Li is a PhD student in the School of Computer Science and Engineering, Beihang University, China. She obtained her Bachelor degree in computer science from Beihang University in 2012. Her research interests include databases, in particular, social data analysis.

Chunming Hu is an associate professor at the School of Computer Science and Engineering, Beihang University, China. He received his PhD degree from Beihang University in 2006. His current research interests include distributed systems, system virtualization, large scale data management and processing systems.

Xuelian Lin is currently a lecturer in the School of Computer Science and Engineering, Beihang University, China. He received his PhD degree from Beihang University in 2013. His current research interests include middleware and data process systems.

Jinpeng Huai is a professor in the School of Computer Science and Engineering at Beihang University, China. He received his PhD in computer science from Beihang University, in 1993. He is an academician of Chinese Academy of Sciences and the vice honorary chairman of China Computer Federation (CCF). His research interests include big data computing, distributed system, virtual computing, service-oriented computing, trustworthiness and security.