Approximate Computation for Big Data Analytics

Shuai Ma, Jinpeng Huai

Abstract—Over the past a few years, research and development has made significant progresses on big data analytics. A fundamental issue for big data analytics is the efficiency. If the optimal solution is unable to attain or not required or has a price to high to pay, it is reasonable to sacrifice optimality with a "good" feasible solution that can be computed efficiently. Existing approximation techniques can be in general classified into approximation algorithms, approximate query processing for aggregate SQL queries and approximation computing for multiple layers of the system stack. In this article, we systematically introduce approximate computation, *i.e.,* query approximation and data approximation, for efficiency and effectiveness big data analytics in practice. We first explain the idea and rationale of query approximation, and show efficiency can be obtained with high effectiveness in practice by its embodiment for graph pattern matching, trajectory compression and dense subgraph computation. We then explain the idea and rationale of data approximation, and show efficiency can be obtained even without sacrificing for effectiveness in practice by its embodiment for shortest paths/distances, network anomaly detection and link prediction.

Index Terms—Big data, query approximation, data approximation

1 Introduction

Over the past a few years, research and development has made significant progresses on big data analytics with the supports from both governments and industries all over the world, such as Spark[1], IBM Watson[2] and Google AlphaGo[3]. A fundamental issue for big data analytics is the efficiency, and various advances towards attacking this issues have been achieved recently, from theory to algorithms to systems [15], [29], [48]. However, *if the optimal solution is unable to attain or not required or has a price to high to pay, it is reasonable to sacrifice optimality with a "good" feasible solution that can be computed efficiently*. Hence, various approximation techniques have been developed, and can in general be classified into three aspects: algorithms, SQL aggregate queries and multiple layers of the system stack.

(1) *Approximation algorithms* were formally defined in the 1970s [20], [28]. An approximation algorithm is necessarily polynomial, and is evaluated by the worst case possible relative error over all possible instances of the NP-hard optimization problem, under the widely believed $P \neq NP$ conjecture. This is relatively mature research field algorithm community, many approximation algorithm have been designed for optimization problems (see books [4], [24], [45]).

(2) *Approximate query processing* supports a slightly constrained set of SQL-style declarative queries, and it specifically provides approximate results for standard SQL aggregate queries, *e.g.,* queries involving COUNT, AVG, SUM and PERCENTILE. Over the past two decades, approximate query processing has been successfully studied, among which sampling technique are heavily employed [9], [21], [30], [42]. Not only traditional DBMS systems, such as Oracle[4], provide approximate functions to support approximate results, but also emerging new systems specially designed for approximate queries, such as BlinkDB[5], Verdict[6], Simba[7], have been designed. However, as pointed out in [9], " it seems impossible to have an AQP system that supports the richness of SQL with significant saving of work while providing an accuracy guarantee that is acceptable to a broad set of application workloads."

(3) *Approximation computing* is a recent computation technique that returns a possibly inaccurate result rather than a guaranteed accurate result from a system point of view. It involves with multiple layers of the system stack from software to hardware to systems (such as approximate circuits, approximate storage and loop perforation), and can be used for applications where an approximate result is sufficient for its purpose [2], [41]. Recently, a workshop on approximate computing across the stack has been usefully held for research on hardware, programming languages and compiler support for approximate computing since 2014. (see *e.g.,* 2016[8], 2017[9] and 2018[10]). Besides the various task oriented quality metrics, the quality-energy trade-off is also concerned for approximate computing. For example, in k-means clustering algorithm, allowing only 5% loss in classification accuracy can provide 50 times energy saving compared with the fully accurate classification [41]

In this article, we present the idea of approximate computation for efficient and effective big data analytics: query approximation and data approximation, based on our recent

• *S. Ma and J. Huai are with the SKLSDE Lab & Beijing Advanced Innovation Center for Big Data and Brain Computing, Beihang University, Beijing, China.*
E-mail: {mashuai, huaijp}@buaa.edu.cn.

1. https://spark.apache.org
2. https://www.ibm.com/watson
3. https://deepmind.com/research/alphago
4. https://oracle-base.com/articles/12c/approximate-query-processing-12cr2
5. http://blindb.org/
6. http://verdictdb.org/
7. https://initialdlab.github.io/Simba/index.html
8. http://approximate.computer/wax2016/
9. http://approximate.computer/wax2017/
10. http://approximate.computer/wax2018/

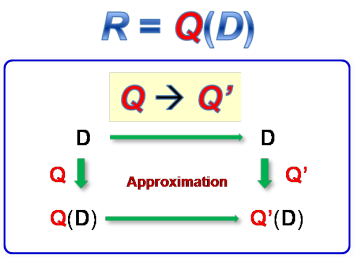

Figure 1. Query approximation

research experiences [13], [14], [25], [31], [33]–[39]. Approximation algorithms ask for feasible solutions that are theoretically bounded with respect to optimal solutions from an algorithm design aspect. Approximate query processing and approximation computing relax the need for accuracy guarantees for aggregate SQL queries and for multiple layers of the system stack, respectively. Similarly, our approximate computation is unnecessarily theoretically bounded with respect to optimal solutions, but from an algorithm design point of view. That is, we focus on approximate computation for big data analytics for a situation where an approximate result is sufficient for a purpose.

2 Query Approximation

Query approximation deals with complex queries involved with big data analytic tasks. Given a class Q of data analytic queries with high a computational complexity, query approximation is to transform into another class Q' of queries with a low computational complexity and satisfiable approximate answers, as depicted in Fig. 1 in which Q, Q', D and R denote the original query, approximate query, data and query result, respectively. Query approximation needs to reach a balance between the query efficiency and answer quality when approximating Q with Q'.

The rationale behind query approximation lies in that inexact or approximate answers are sufficient or acceptable for many big data analytic tasks. On one hand, when the volume of data is extremely large, it may be impossible or not necessary to compute the exact answers. Observe that nobody would try each and every store to find a pair of shoes with the best cost-performance ratio. That is, inexact (approximate) solutions are good enough for certain cases. On the other hand, when taking noises (very common for big data) into account, it may not always be a good idea to compute exact answers for those data analytic tasks whose answers are rare or hard to identify, such as the detection of homegrown violent extremists (HVEs) who seek to commit acts of terrorism in the United States and abroad [26], as exact solutions may have a high chance to miss/ignore possible candidates.

We next explain query approximation computation in more detail using three different data analytic tasks.

(1) Strong Simulation [33], [34]. Given a pattern graph Q and a data graph G, *graph pattern matching* is to find all subgraphs of G that match Q, and is being increasingly used in various applications, *e.g.,* biology and social networks.

Here *matching* is typically defined in terms of *subgraph isomorphism* [19]: a subgraph G_s of G *matches* Q if there exists a *bijective function* f from the nodes of Q to the nodes in G_s such that (a) for each pattern node u in Q, u and $f(u)$ have the same label, and (b) there exists an edge (u, u') in Q if and only if there exists an edge $(f(u), f(u'))$ in G_s.

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. However, subgraph isomorphism is NP-complete, and may return exponential many matched subgraphs. Further, subgraph isomorphism is too restrictive to find sensible matches in certain scenarios, as observed in [17]. Even worse, online data in many cases only represents a partial world (*e.g.,* terrorist collaboration networks and homosexual networks often involve with a large amount of off-line data). Exact computations on such online data, whose off-line counterpart is extremely hard to gather, typically decreases the chance of identifying candidate answers. These hinder the usability of graph pattern matching in emerging applications.

To lower the high complexity of subgraph isomorphism, substitutes for subgraph isomorphism [16], [17], which allow graph pattern matching to be conducted in cubic-time, have been proposed by extending graph simulation [23]. However, they fall short of capturing the topology of data graphs, i.e., graphs may have a structure drastically different from pattern graphs that they match, and the matches found are often too large to analyze.

To rectify these problems, strong simulation, an "approximate" substitute for subgraph isomorphism, is proposed for graph pattern matching [34], which (a) theoretically preserves the key topology of pattern graphs and finds a bounded number of matches, (b) retains the same complexity as earlier extensions of graph simulation [16], [17], by providing a cubic-time algorithm for strong simulation compuation, and (c) has the locality property that allows us to develop an effective distributed algorithm to conduct graph pattern matching on distributed graphs.

Strong simulation is experimentally verified that it is able to identify sensible matches that are not found by subgraph isomorphism, and it finds high quality matches that retain graph topology. Indeed, 70%-80% of matches found by subgraph isomorphism are retrieved by strong simulation. Further, strong simulation is over 100 times faster than subgraph isomorphism, and has a bounded number of matches.

(2) One-Pass Trajectory Compression [31]. Trajectory compression (*a.k.a.* trajectory simplification) is to compress data points in a trajectory to a set of continuous line segments, and is commonly used in practice.

The compression ratios of lossless methods are poor, and querying on the compressed data is time consuming due to the reconstruction of the original data [43]. Hence, lossy techniques, which provide approximate solutions with good compression ratios and bounded errors, are the mainstream.

Piece-wise line simplification (LS) comes from the computational geometry, whose target is to approximate a given finer piece-wise linear curve by another coarser piece-wise linear curve (normally a subset of the former), such that the

maximum distance of the former from the later is bounded by a user specified constant (*i.e.,* error bound). It is widely used due to its distinct advantages: (a) simple and easy to implement, (b) no need extra knowledge and suitable for freely moving objects, and (c) bounded errors with good compression ratios.

LS algorithms fall into two categories: *optimal* and *approximate*. Optimal methods [27] are to find the minimum number of points or segments to represent the original polygonal lines *w.r.t.* an error bound ϵ. They have higher time and space complexities, and are not practical for large trajectory data. Hence, various approximate LS algorithms have been developed, from batch algorithms (*e.g.,* [12]) to online algorithms (*e.g.,* [32]) and to one-pass algorithms (*e.g.,* [31]).

An LS algorithm is *one-pass* if it processes each point in a trajectory once and only once when compressing the trajectory. Obviously, one-pass algorithms have low time and space complexities, and are more appropriate for online processing. The difficulty comes from the need to achieve effective compression ratios. Existing trajectory simplification algorithms (*e.g.,* [12]) and online algorithms (*e.g.,* [32]) essentially employ global distance checking, although online algorithms restrict the checking within a window. That is, whenever a new line segment is formed, these algorithms always check its distances to all or a subset of data points, and, therefore, a data point is checked multiple times, depending on its order in the trajectory and the number of directed line segments formed. Hence, *an appropriate local distance checking approach is needed in the first place for one-pass trajectory simplification.*.

We develop a local distance checking method, referred to as *fitting function*, such that a data point is checked only once in the entire process. Based on the fitting function, we develop one-pass error bounded trajectory simplification algorithms OPERB and OPERB-A that scan each data point in a trajectory once and only once, allowing interpolating new data points or not, respectively. By comparing our algorithms with FBQS (the fastest existing LS online algorithm [32]) and DP (the best existing LS batch algorithm in terms of compression ratio [12]), our one-pass algorithms OPERB and OPERB-A are over four times faster than FBQS, and have comparable compression ratios with DP.

(3) Dense Temporal Subgraph Computation [39]. We study dense subgraphs in *a special type of temporal networks* whose nodes and edges are kept fixed, but edge weights constantly and regularly vary with timestamps [39]. Essentially, a temporal network with T timestamps can be viewed as T snapshots of a static network such that the network nodes and edges are kept the same among these T snapshots, while the edge weights vary with network snapthots. Road traffic networks typically fall into this category of temporal networks, and dense subgraphs are used for road traffic analyses that are of particular importance for transportation management of large cities.

Dense subgraphs are a general concept, and their concrete semantics highly depend on the studied problems and applications. Though dense subgraphs have been widely studied in static networks, how to properly define their semantics over temporal networks is still in the early stage, not to mention effective and efficient analytic algorithms.

We adopt the *form of dense temporal subgraphs* initially defined and studied in [5], such that a temporal subgraph corresponds to a connected subgraph measured by the sum of all its edge weights in a time interval, *i.e.,* a continuous sequence of timestamps. Intuitively, a dense subgraph that we consider corresponds to a collection of connected highly slow or jam roads (*i.e.,* a jam area) in road networks, lasting for a continuous sequence of snapshots.

The problem of finding dense subgraphs in temporal networks is non-trivial, and it is already NP-complete even for a temporal network with a single snapshot and with $+1$ or -1 edge weights only, as observed in [5]. Even worse, it remains hard to approximate for temporal networks with single snapshots [39]. Moreover, given a temporal network with T timestamps, there are a total number of $T*(T+1)/2$ time intervals to consider, which further aggravates the difficulty. The state of the art solution MEDEN [5] adopts a Filter-And-Verification (FAV) framework that *even if a large portion of time intervals are filtered, there often remain a large number of time intervals to verify*. Hence, this method is not big data friendly, and is not scalable when temporal networks have a large number of nodes/edges or a large number T of timestamps.

We develop a data-driven approach (referred to as FIDES), instead of filter-and-verification, to identifying the most possible k time intervals from $T \times (T+1)/2$ time intervals, in which T is the number of snapshots and k is a small constant, *e.g.,* 10. This is achieved by exploring the characteristics of time intervals involved with dense subgraphs based on the observation of *evolving convergence phenomenon* in traffic data, inspired by the convergent evolution in nature[11]. That is, our method provides time intervals with probabilistic guarantees, instead of exact ones as FAV. Using both real-life and synthetic data, we experimentally show that our method FIDES is over 1000 times faster than MEDEN [5], while the quality of dense subgraphs found is comparable with MEDEN .

3 Data Approximation

Big data has a large volume, and, hence, the space complexity [11] of big data analytic tasks starts raising more concerns. Given a class Q of queries on data D, data approximation is to transform D into smaller D' such that Q on D' returns a sufficient or satisfiable approximate answer in a more efficient way. Further, it is typically common that query Q needs to be (slightly) modified to Q' to accommodate the changes of D to D', as shown in Fig. 2. Similar to query approximation, data approximation needs to reach a balance between the query efficiency and answer quality.

The rationale behind data approximation has roots in the Pareto principle[12] that "states that, for many events, roughly 80% of the effects come from 20% of the causes". The critical thing for data approximation is to determine which part of data is relevant to tasks (belong to the 20%). By this principle, for many big analytic tasks, one may only need to keep a small amount of the data to derive high quality

11. https://en.wikipedia.org/wiki/Convergent_evolution
12. https://en.wikipedia.org/wiki/Pareto_principle

answers. For example, when we are to build a predictive model on the stock of razers for an online store based on the order history of customers, orders from men are good enough. While on the stock of lipsticks, those from women are good enough. That is to say, it is not necessary to use the entire data for certain data analytic tasks.

However, it should be pointed out that there are data analytic tasks such that data approximation could not work well. For example, an online store needs to count the total number of goods in its catalog. Essentially entire goods should be considered for this task, and if a (small) portion of goods are chosen, it is hard to have a satisfiable result.

We next explain data approximation computation in more detail using three different data analytic tasks.

(1) Proxies for Shortest Paths and Distances [36], [37]. Computing shortest paths and distances is one of the fundamental problems on graphs. We study the *node-to-node shortest path* (*distance*) problem on large graphs: given a weighted undirected graph $G(V, E)$ with non-negative edge weights, and two nodes of G, the source s and the target t, find the shortest path (distance) from s to t in G. The Dijkstra's algorithm with Fibonacci heaps runs in $O(n \log n+m)$ due to Fredman & Tarjan [11], where n and m denote the numbers of nodes and edges in a graph, respectively, which remains asymptotically the fastest known solution on arbitrary undirected graphs with non-negative edge weights. However, computing shortest paths and distances remains a challenging problem, in terms of both time and space cost, on large-scale graphs. Hence, various optimizations have been developed to speed-up the computation.

To speed-up shortest path and distance queries, we propose *proxies* that have the following properties: (a) each proxy captures a set of nodes in a graph, referred to as DRA, (b) a small number of proxies can represent a large number of nodes in a graph, (c) shortest paths and distances involved within the set of nodes being represented by the same proxies can be answered efficiently, and, (d) the proxies and the set of nodes being represented can be computed efficiently in *linear* time.

The framework for speeding-up shortest path and distance queries with proxies consists of modules preprocessing and query answering as follows.
(a) *Preprocessing*: Given graph $G(V, E)$, it first computes all DRAs and their maximal proxies in linear time, then it computes and stores all the shortest paths and distances between any node and its proxy. It finally computes the reduced subgraph G' by removing all DRAs from graph G, *i.e.,* keeping the proxies only.
(b) *Query answering*. Given two nodes s and t in graph $G(V, E)$ and the pre-computed information, the query answering module essentially executes the following.

The shortest path $path(s, t) = path(s, u_s)/\ path(u_s, u_t)/\ path(u_t, t)$, where u_s and u_t are the proxies of s and t, respectively. As $path(s, u_s)$ and $path(u_t, t)$ are pre-computed, and $path(u_s, u_t)$ can be computed on the reduced subgraph G' by invoking any existing algorithms (*e.g.,*AH [49]). The shortest distance $dist(s, t) = dist(s, u_s) + dist(u_s, u_t) + dist(u_t, t)$ can be computed along the same line.

Essentially, we propose a light-weight data reduction technique for speeding-up (exact) shortest path and distance queries on large weighted undirected graphs [36]. We experimentally show that about $1/3$ nodes of real-life social and road networks are captured by proxies.

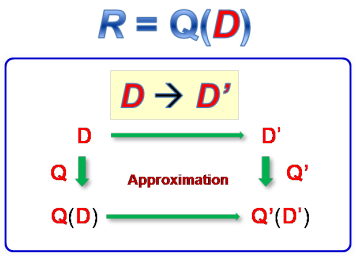

Figure 2. Data approximation

(2) Network Anomaly Detection [25] Anomaly (or outlier) detection aims at identifying those objects in a database that are unusual, i.e., different than the majority of the data and therefore suspicious resulting from a contamination, error, or fraud [50]. Network anomaly detection has become very popular recently because of the importance of discovering key regions of structural inconsistency in the network. In addition to application-specific information carried by anomalies, the presence of such structural inconsistency is often an impediment to the effective application of data mining algorithms such as community detection and classification.

Networks are inherently complex entities, and, hence, anomalies may be defined in a wide variety of ways. Our goal is to discover *structural inconsistencies, i.e., the anomalous nodes that connect to a number of diverse influential communities*, inspired by the concept of social brokers across groups, which provide social capital in networks [7]. While a variety of graph embeddings, such as multidimensional scaling [6], are available in the literature, they aim to preserve (global) pairwise similarities and are not optimized to networks and the problem of anomaly detection. Hence, they cannot be directly used for the detection of structural inconsistencies proposed in this paper. We propose a novel graph embedding method, specifically designed to ferret out the anomalous nodes in large networks.

Our embedding approach is based on a model, in which each dimension of the embedding corresponds to a clustered region in the network. In other words, the similarity of different nodes along a particular dimension, indicates their similarity to a particular clustered region. Therefore, this embedding retains a very high level of interpretability in terms of the original graph data, which is very useful from an application-specific perspective. The nature of the embedding also makes it possible to detect anomalous nodes, by examining the interaction of each node with the different regions in terms of the embedding. In particular, we measure the *level of anomalousness* of a node in terms of the embedding imposed on the node and its neighbors.

Dimensionality of embedding, each node is represented as a d-dimensional vector, can be large. Hence, such an approach is rather hard to apply to the case of large networks, because the complexity of the approach is in proportion to the square of the number of nodes when optimizing the embedding and because the noises in the embedding seriously impair the accuracy of detected anomalous nodes.

Hence, we incorporate data approximation techniques (sampling, graph partitioning techniques, and, moreover, a novel dimension reduction technique) to make the approach more scalable and effective for large networks.

Essentially, in our graph embedding, d represents the number of communities. As the anomalous nodes are only determined by influential communities and nodes typically connect to a limited number of communities, the complete d-dimensions are unnecessary, and a limited number of communities suffice to ascertain anomalies. Thus, our dimension reduction technique (referred to as $k+\beta$ reduction) only maintains $(k+\beta)$-dimensions for embedding of each node, where k is the maximum number of communities to connect, β is to tolerate mistakes when determining the k communities and is removed after the computation process. Here $k, \beta \ll d$, *e.g.*, $k = 10$ and $\beta = 2$ and $d = 600$ for a network with 10^6 nodes.

Using both real-life data and synthetic data , we conduct an extensive experimental study. (a) The modularity [10] was increased about 4.9% and 3.6% with our approach and OddBall [3], respectively; (c) Our approach scales to graph graphs with large number of communities, while traditional multidimensional scaling approach [6] ran out of memory.

(3) Ensemble Enabled Link Prediction [13], [14]. Link prediction is the task to predict the formation of future links in a dynamic and evolving network, and has been extensively studied due to its numerous applications, such as the recommendation of friends in a social network, images in a multimedia network, or collaborators in a scientific network.

Link prediction methods are often applied to very large and sparse networks, which have a large *search space* $O(n^2)$, where n is the number of nodes. Hence, the scalability is a big challenge. In fact, an often overlooked fact is that most *exiting link prediction algorithms evaluate the link propensities only over a subset of possibilities rather than all propensities over the entire network*. Consider a large network with 10^8 nodes. Its number of *possibilities* for links is of the order of 10^{16}. Therefore, a 3GHz processor would require at least 35 days just to allocate one *machine cycle* to every pair of nodes. This implies that in order to determine the top-ranked link predictions over the *entire network*, the running time would be much more than 35 days.

It is noteworthy that most existing link prediction algorithms are not designed to search over the entire $O(n^2)$ possibilities. A closer examination of the relevant studies shows that even for networks of modest size, these algorithms perform benchmark evaluations over a *sample of the possibilities* for links. In other words, the *complete ranking problem for link prediction in very large networks remains challenging at least from a computational point of view*.

Latent factor models have proven a great success for collaborative filtering, but not link prediction in spite of the obvious similarity and the obvious effectiveness of latent factor models. One of the reasons why latent factor models are rarely used for link prediction is due to their complexity. In collaborative filtering applications, the item dimension is of the order of a few hundred thousand, whereas even the smallest real-world networks contain more than a million nodes. Even worse, we also experientially find that the quality of link prediction for latent factor models decreases with the increase of data sparsity, and networks typically become sparser when their sizes grow larger.

We explore an *ensemble approach* to making latent factor models practical for link prediction by decomposing the search space into a set of smaller matrices with three structural bagging methods with performance guarantees, which has obvious *effectiveness* advantages. In this way, latent factor models only need to deal with networks with small sizes (and denser), and retain their effectiveness and efficiency. By incorporating with the characteristics of link prediction, the bagging methods maintain high prediction accuracy while reducing the network size via graph sampling techniques. Further, the use of an ensemble approach has obvious robustness advantages as well.

We experimentally show that our ensemble approach is over 50 times faster and over 20% more accurate than BIGCLAM [47] using real-life social networks.

4 Beyond Approximation Techniques

For big data analytics, there are no one-size-fits-all techniques, and it is often necessary to combine different techniques to obtain good solutions.

We have seen that sampling helps to achieve a balance between efficiency and effectiveness for approximate query processing [9], [21], [30], [42] and link prediction [13], [14], and other techniques such as incremental computation [17], [38], [44], distributed computing [18], [35], and system techniques *e.g.*, caching [46], hardware [1], [22] can also be unitized for big data analytics, and can even be combined for designing query and data approximation techniques for big data analytics.

It is worth pointing out that for all kind of techniques big data analytics, various computing resources should be seriously considered, *e.g.*, using bounded resources for approximation [8] and for incremental computation [40]. It is also obvious that techniques beyond query approximation and data approximation, such as theory and systems [15], [29], [48], are must things for big data analytics.

5 Conclusions

In this article we have systematiclly introduced approximation computation techniques for efficient and effective big data analytics. Furthermore, although approximate computation does not put theoretical bounds with respect to optimal solutions, it does expect a balance between efficiency and effectiveness. Indeed, (a) our query approximation techniques [31], [33], [34], [39] show that efficiency can be obtained with high accuracy in practice, and (b) our data approximation techniques for shortest paths/distances and link prediction [13], [14], [25], [36], [37] show that efficiency and accuracy can be obtained simultaneously for certain data analytic tasks. That is, the design policy of approximate computation is not to sacrifice effectiveness for efficiency, but to achieve both efficiency and effectiveness in practice.

Acknowledgement. This work is supported in part by 973 program (2014CB340300), NSFC (U1636210 & 61421003). We would also thank our colleagues Charu Aggarwal, Gao Cong, Wenfei Fan, Chunming Hu, Xuelian Lin, Haixun Wang, Tianyu Wo and our students Yang Cao, Liang Duan, Kaiyu Feng, Renjun Hu, Han Zhang for their joined efforts.

REFERENCES

[1] C. R. Aberger, A. Lamb, S. Tu, A. Nötzli, K. Olukotun, and C. Ré. Emptyheaded: A relational engine for graph processing. *ACM Trans. Database Syst.*, 42(4):20:1–20:44, 2017.

[2] A. Agrawal, J. Choi, K. Gopalakrishnan, S. Gupta, R. Nair, J. Oh, D. A. Prener, S. Shukla, V. Srinivasan, and Z. Sura. Approximate computing: Challenges and opportunities. In *IEEE International Conference on Rebooting Computing, ICRC*, 2016.

[3] L. Akoglu, H. Tong, and D. Koutra. Graph based anomaly detection and description: a survey. *Data Min. Knowl. Discov.*, 29(3):626–688, 2015.

[4] G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, Marchetti-Spaccamela, and M. A., Protasi. *Complexity and Approximation: Combinatorial Optimization Problems and Their Approximability Properties*. Springer, 1999.

[5] P. Bogdanov, M. Mongiovì, and A. K. Singh. Mining heavy subgraphs in time-evolving networks. In *ICDM*, 2011.

[6] I. Borg and P. Groenen. *Modern Multidimensional Scaling: Theory and Applications (2nd ed.)*. Springer, 2005.

[7] R. S. Burt. Structural holes and good ideas. *American Journal of Sociology*, 110(2):349–399, 2004.

[8] Y. Cao and W. Fan. Data driven approximation with bounded resources. *PVLDB*, 10(9):973–984, 2017.

[9] S. Chaudhuri, B. Ding, and S. Kandula. Approximate query processing: No silver bullet. In *SIGMOD*, 2017.

[10] A. Clauset, M. E. J. Newman, and C. Moore. Finding community structure in very large networks. *Physical Review E*, 70:066111, 2004.

[11] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. The MIT Press, 2001.

[12] D. H. Douglas and T. K. Peucker. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *The Canadian Cartographer*, 10(2):112–122, 1973.

[13] L. Duan, C. C. Aggarwal, S. Ma, R. Hu, and J. Huai. Scaling up link prediction with ensembles. In *WSDM*, 2016.

[14] L. Duan, S. Ma, C. Aggarwal, T. Ma, and J. Huai. An ensemble approach to link prediction. *TKDE*, 29(11):2402–2416, 2017.

[15] W. Fan, F. Geerts, and F. Neven. Making queries tractable on big data with preprocessing. *PVLDB*, 6(9):685–696, 2013.

[16] W. Fan, J. Li, S. Ma, N. Tang, and Y. Wu. Adding regular expressions to graph reachability and pattern queries. In *ICDE*, 2011.

[17] W. Fan, J. Li, S. Ma, N. Tang, Y. Wu, and Y. Wu. Graph pattern matching: From intractable to polynomial time. *PVLDB*, 3(1), 2010.

[18] W. Fan, J. Xu, Y. Wu, W. Yu, J. Jiang, Z. Zheng, B. Zhang, Y. Cao, and C. Tian. Parallelizing sequential graph computations. In *SIGMOD*, 2017.

[19] B. Gallagher. Matching structure and semantics: A survey on graph-based pattern matching. *AAAI FS.*, 2006.

[20] M. R. Garey, R. L. Graham, and J. D. Ullman. Worst-case analysis of memory allocation algorithms. In *STOC*, 1972.

[21] M. N. Garofalakis and P. B. Gibbons. Approximate query processing: Taming the terabytes. In *VLDB*, 2001.

[22] S. Han, L. Zou, and J. X. Yu. Speeding up set intersections in graph algorithms using SIMD instructions. In *SIGMOD*, 2018.

[23] M. R. Henzinger, T. A. Henzinger, and P. W. Kopke. Computing simulations on finite and infinite graphs. In *FOCS*, 1995.

[24] D. S. Hochbaum. *Approximation Algorithms for NP-Hard Problems*. Springer, 1996.

[25] R. Hu, C. C. Aggarwal, S. Ma, and J. Huai. An embedding approach to anomaly detection. In *ICDE*, 2016.

[26] B. W. K. Hung and A. P. Jayasumana. Investigative simulation: Towards utilizing graph pattern matching for investigative search. In *ASONAM*, 2016.

[27] H. Imai and M. Iri. Computational-geometric methods for polygonal approximations of a curve. *Computer Vision, Graphics, and Image Processing*, 36:31–41, 1986.

[28] D. S. Johnson. Approximation algorithms for combinatorial problems. *J. Comput. Syst. Sci.*, 9(3):256–278, 1974.

[29] M. I. Jordan. Computational thinking, inferential thinking and "big data". In *PODS*, 2015.

[30] T. Kraska. Approximate query processing for interactive data science. In *SIGMOD*, 2017.

[31] X. Lin, S. Ma, H. Zhang, T. Wo, and J. Huai. One-pass error bounded trajectory simplification. *PVLDB*, 10(7):841–852, 2017.

[32] J. Liu, K. Zhao, P. Sommer, S. Shang, B. Kusy, and R. Jurdak. Bounded quadrant system: Error-bounded trajectory compression on the go. In *ICDE*, 2015.

[33] S. Ma, Y. Cao, W. Fan, J. Huai, and T. Wo. Capturing topology in graph pattern matching. *PVLDB*, 5(4):310–321, 2011.

[34] S. Ma, Y. Cao, W. Fan, J. Huai, and T. Wo. Strong simulation: Capturing topology in graph pattern matching. *TODS*, 39(1):4:1–4:46, 2014.

[35] S. Ma, Y. Cao, J. Huai, and T. Wo. Distributed graph pattern matching. In *WWW*, 2012.

[36] S. Ma, K. Feng, J. Li, H. Wang, G. Cong, and J. Huai. Proxies for shortest path and distance queries. *TKDE*, 28(7):1835–1850, 2016.

[37] S. Ma, K. Feng, J. Li, H. Wang, G. Cong, and J. Huai. Proxies for shortest path and distance queries. In *IEEE*, 2017.

[38] S. Ma, C. Gong, R. Hu, D. Luo, C. Hu, and J. Huai. Query independent scholarly article ranking. In *ICDE*, 2018.

[39] S. Ma, R. Hu, L. Wang, X. Lin, and J. Huai. Fast computation of dense temporal subgraphs. In *ICDE*, 2017.

[40] S. Ma, J. Li, C. Hu, X. Liu, and J. Huai. Graph pattern matching for dynamic team formation. *CoRR*, abs/1801.01012, 2018.

[41] S. Mittal. A survey of techniques for approximate computing. *ACM Comput. Surv.*, 48(4):62:1–62:33, 2016.

[42] B. Mozafari. Approximate query engines: Commercial challenges and research opportunities. In *SIGMOD*, 2017.

[43] A. Nibali and Z. He. Trajic: An effective compression system for trajectory data. *TKDE*, 27(11):3138–3151, 2015.

[44] G. Ramalingam and T. Reps. On the computational complexity of dynamic graph problems. *TCS*, 158(1-2), 1996.

[45] V. V. Vazirani. *Approximation Algorithms*. Springer, 2003.

[46] J. Wang, Z. Liu, S. Ma, N. Ntarmos, and P. Triantafillou. GC: A graph caching system for subgraph/supergraph queries. *PVLDB*, 11(12):2022–2025, 2018.

[47] J. Yang and J. Leskovec. Overlapping community detection at scale: A nonnegative matrix factorization approach. In *WSDM*, 2013.

[48] M. Zaharia, R. S. Xin, P. Wendell, T. Das, M. Armbrust, A. Dave, X. Meng, J. Rosen, S. Venkataraman, M. J. Franklin, A. Ghodsi, J. Gonzalez, S. Shenker, and I. Stoica. Apache spark: a unified engine for big data processing. *Commun. ACM*, 59(11):56–65, 2016.

[49] A. D. Zhu, H. Ma, X. Xiao, S. Luo, Y. Tang, and S. Zhou. Shortest path and distance queries on road networks: towards bridging theory and practice. In *SIGMOD*, 2013.

[50] A. Zimek and E. Schubert. *Outlier Detection*, pages 1–5. Springer New York, 2017.

PLACE PHOTO HERE

Shuai Ma is a professor at the School of Computer Science and Engineering, Beihang University, China. He obtained his PhD degrees from University of Edinburgh in 2010, and from Peking University in 2004, respectively. He was a postdoctoral research fellow in the database group, University of Edinburgh, a summer intern at Bell labs, Murray Hill, USA and a visiting researcher of MRSA. He is a recipient of the best paper award for VLDB 2010 and the best challenge paper award for WISE 2013. His current research interests include database theory and systems, social data and graph analysis, and data intensive computing.

PLACE PHOTO HERE

Jinpeng Huai is a professor at the School of Computer Science and Engineering, Beihang University, China. He received his Ph.D. degree in computer science from Beihang University, China, in 1993. Prof. Huai 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 systems, virtual computing, service-oriented computing, trustworthiness and security.