Approximate Computation for Big Data Analytics

Shuai Ma AND Jinpeng Huai
SKLSDE Lab, Beihang University, China
Beijing Advanced Innovation Center for Big Data and Brain Computing, China

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 unnecessary 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 efficient and effective big data analytics. We explain the ideas and rationales behind query and data approximation, and show efficiency can be obtained with high effectiveness, and even without sacrificing for effectiveness, for certain data analytic tasks.

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, IBM Watson and Google AlphaGo. A fundamental issue for big data analytics is the efficiency, and various advances towards attacking this issue have been achieved recently, from theory to algorithms to systems [Fan et al. 2013; Jordan 2015; Zaharia et al. 2016]. 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 [Garey et al. 1972; Johnson 1974]. 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 [Vazirani 2003]).
(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 [Chaudhuri et al. 2017; Mozafari 2017]. Not only traditional DBMS systems, such as Oracle, provide approximate functions to support approximate results, but also emerging new systems specially designed for approximate queries, such as BlinkDB, Verdict and Simba, have been designed. Note that, as pointed out in [Chaudhuri et al. 2017], " it seems impossible to have an approximate query processing 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 [Agrawal et al. 2016; Mittal 2016]. 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. Besides the various task oriented quality metrics, the quality-energy trade-off is also concerned for approximate computing. For instance, allowing only 5% loss of classification accuracy can provide 50 times energy saving for clustering algorithm k-means [Mittal 2016].

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 research experiences [Ma et al. 2014; Ma et al. 2020; Ma et al. 2016; Ma et al. 2016; Duan et al. 2017]. 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. 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 [Hung and Jayasumana 2016], as finding exact solutions may have a high chance to miss/ignore possible candidates.

We next explain query approximation in more detail using two data analytic tasks.

(1) Strong Simulation [Ma et al. 2014]. 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* [Gallagher 2006]: 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 is an edge (u, u') in Q if and only if there is 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 exponentially many matched subgraphs. Further, subgraph isomorphism is too restrictive to find sensible matches in certain scenarios, as observed in [Fan et al. 2010]. Even worse, online data in many cases only represents a partial world (*e.g.,* terrorist collaboration networks and homosexual networks are often accompanied with a large amount of off-line data). Exact computations on online data, whose off-line counterpart is extremely hard to collect, 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 [Fan et al. 2010; Fan et al. 2011], which allow graph pattern matching to be conducted in cubic-time, have been proposed by extending graph simulation [Henzinger et al. 1995]. 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, 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 [Fan et al. 2010; Fan et al. 2011], by providing a cubic-time algorithm for 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.

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) Dense Temporal Subgraph Computation [Ma et al. 2020]. Dense subgraphs in *a special type of temporal networks* are studied, whose nodes and edges are kept fixed, but edge weights constantly and regularly vary with timestamps. 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 snapshots. 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 depends 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 algorithms. We adopt the *form of dense temporal subgraphs* initially defined and studied in [Bogdanov et al. 2011], such that a temporal subgraph is 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 [Bogdanov et al. 2011]. Even worse, it remains hard to approximate for temporal networks with single snapshots. 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 [Bogdanov et al. 2011] 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 timestamps.

We develop a data-driven approach (referred to as FIDES), instead of FAV, 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[1]. 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 [Bogdanov et al. 2011], 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 [Cormen et al. 2001] 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'. Similar to query approximation, data approximation needs to reach a balance between the query efficiency and answer quality.

[1] https://en.wikipedia.org/wiki/Convergent_evolution

The rationale behind data approximation has roots in the Pareto principle[2] 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 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. 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 in more detail using two data analytic tasks.

(1) Proxies for Shortest Paths and Distances [Ma et al. 2016]. 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 [Cormen et al. 2001], 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 two module, 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 only $path(u_s, u_t)$ needs to be computed on the reduced subgraph G' by invoking any existing algorithms (*e.g.,*AH [Zhu et al. 2013]). 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.

[2]https://en.wikipedia.org/wiki/Pareto_principle

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

(2) Ensemble Enabled Link Prediction [Duan et al. 2017]. Link prediction is the task to predict the formation of future links in dynamic and evolving networks, and has been extensively studied due to its various 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, they perform benchmark evaluations over a *sample of the possibilities* for links. That is, 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, items have a few hundred thousand dimensions, whereas even the smallest real-world networks contain more than a million nodes. Even worse, we also experientially verify 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 [Yang and Leskovec 2013] using real-life social networks.

4. CONCLUSIONS

In this article we have systematically introduced approximation computation techniques for efficient and effective big data analytics. Furthermore, although approximate compu-

tation 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 show that efficiency can be obtained with high accuracy in practice, and (b) our data approximation techniques show that efficiency and accuracy can be obtained simultaneously for certain data analytic tasks. That is, though approximate computation is for a situation where an approximate result is sufficient for a purpose, its design policy is not always to sacrifice effectiveness for efficiency.

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 [Chaudhuri et al. 2017; Mozafari 2017] and link prediction [Duan et al. 2017], and other techniques such as machine learning [Bengio et al. line], incremental computation [Ramalingam and Reps 1996; Fan et al. 2010; Ma et al. 2018], distributed computing [Ma et al. 2012], and system techniques (*e.g.,* caching [Wang et al. 2018], hardware [Aberger et al. 2017]) can also be unitized when designing query and data approximation techniques for big data analytics.

It is also worth pointing out that (a) for all kind of techniques big data analytics, various computing resources should be seriously considered, *e.g.,* using bounded resources for approximation [Cao and Fan 2017] and for incremental computation [Ma et al. 2018], and (b) theoretical analyses are also important for developing approximation techniques. For instance, our query and data approximation techniques are based serious theoretical results [Ma et al. 2014; Ma et al. 2016; Ma et al. 2020].

ACKNOWLEDGMENTS

This work is supported in part by NSFC 61925203 & U1636210 & 61421003. We would also thank our colleagues Charu Aggarwal, Wenfei Fan and our students Yang Cao, Liang Duan, Kaiyu Feng, Renjun Hu, Jia Li for their joined efforts.

REFERENCES

ABERGER, C. R., LAMB, A., TU, S., NÖTZLI, A., OLUKOTUN, K., AND RÉ, C. 2017. Emptyheaded: A relational engine for graph processing. *ACM Trans. Database Syst. 42,* 4, 20:1–20:44.

AGRAWAL, A., CHOI, J., GOPALAKRISHNAN, K., GUPTA, S., NAIR, R., OH, J., PRENER, D. A., SHUKLA, S., SRINIVASAN, V., AND SURA, Z. 2016. Approximate computing: Challenges and opportunities. In *ICRC*.

BENGIO, Y., LODI, A., AND PROUVOST, A. 2020 (Available online). Machine learning for combinatorial optimization: A methodological tour dhorizon. *European Journal of Operational Research*.

BOGDANOV, P., MONGIOVÌ, M., AND SINGH, A. K. 2011. Mining heavy subgraphs in time-evolving networks. In *ICDM*.

CAO, Y. AND FAN, W. 2017. Data driven approximation with bounded resources. *PVLDB 10,* 9, 973–984.

CHAUDHURI, S., DING, B., AND KANDULA, S. 2017. Approximate query processing: No silver bullet. In *SIGMOD*.

CORMEN, T. H., LEISERSON, C. E., RIVEST, R. L., AND STEIN, C. 2001. *Introduction to Algorithms*. The MIT Press.

DUAN, L., MA, S., AGGARWAL, C., MA, T., AND HUAI, J. 2017. An ensemble approach to link prediction. *IEEE Trans. Knowl. Data Eng. 29,* 11, 2402–2416.

FAN, W., GEERTS, F., AND NEVEN, F. 2013. Making queries tractable on big data with preprocessing. *PVLDB 6,* 9, 685–696.

FAN, W., LI, J., MA, S., TANG, N., AND WU, Y. 2011. Adding regular expressions to graph reachability and pattern queries. In *ICDE*.

FAN, W., LI, J., MA, S., TANG, N., WU, Y., AND WU, Y. 2010. Graph pattern matching: From intractable to polynomial time. *PVLDB 3,* 1, 264–275.

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

GAREY, M. R., GRAHAM, R. L., AND ULLMAN, J. D. 1972. Worst-case analysis of memory allocation algorithms. In *STOC*.

HENZINGER, M. R., HENZINGER, T. A., AND KOPKE, P. W. 1995. Computing simulations on finite and infinite graphs. In *FOCS*.

HUNG, B. W. K. AND JAYASUMANA, A. P. 2016. Investigative simulation: Towards utilizing graph pattern matching for investigative search. In *ASONAM*.

JOHNSON, D. S. 1974. Approximation algorithms for combinatorial problems. *JCSS 9,* 3, 256–278.

JORDAN, M. I. 2015. Computational thinking, inferential thinking and "big data". In *PODS*.

MA, S., CAO, Y., FAN, W., HUAI, J., AND WO, T. 2014. Strong simulation: Capturing topology in graph pattern matching. *ACM Trans. Database Syst. 39,* 1, 4:1–4:46.

MA, S., CAO, Y., HUAI, J., AND WO, T. 2012. Distributed graph pattern matching. In *WWW*.

MA, S., FENG, K., LI, J., WANG, H., CONG, G., AND HUAI, J. 2016. Proxies for shortest path and distance queries. *IEEE Trans. Knowl. Data Eng. 28,* 7, 1835–1850.

MA, S., HU, R., WANG, L., LIN, X., AND HUAI, J. 2020. An efficient approach to finding dense temporal subgraphs. *IEEE Trans. Knowl. Data Eng. 32,* 4, 645–658.

MA, S., LI, J., HU, C., LIN, X., AND HUAI, J. 2016. Big graph search: challenges and techniques. *Frontiers Comput. Sci. 10,* 3, 387–398.

MA, S., LI, J., HU, C., LIU, X., AND HUAI, J. 2018. Graph pattern matching for dynamic team formation. *CoRR abs/1801.01012*.

MITTAL, S. 2016. A survey of techniques for approximate computing. *ACM Comput. Surv. 48,* 4, 62:1–62:33.

MOZAFARI, B. 2017. Approximate query engines: Commercial challenges and research opportunities. In *SIGMOD*.

RAMALINGAM, G. AND REPS, T. W. 1996. On the computational complexity of dynamic graph problems. *Theor. Comput. Sci. 158,* 1&2, 233–277.

VAZIRANI, V. V. 2003. *Approximation Algorithms*. Springer.

WANG, J., LIU, Z., MA, S., NTARMOS, N., AND TRIANTAFILLOU, P. 2018. GC: A graph caching system for subgraph/supergraph queries. *PVLDB 11,* 12, 2022–2025.

YANG, J. AND LESKOVEC, J. 2013. Overlapping community detection at scale: A nonnegative matrix factorization approach. In *WSDM*.

ZAHARIA, M., XIN, R. S., WENDELL, P., DAS, T., ARMBRUST, M., DAVE, A., MENG, X., ROSEN, J., VENKATARAMAN, S., FRANKLIN, M. J., GHODSI, A., GONZALEZ, J., SHENKER, S., AND STOICA, I. 2016. Apache spark: a unified engine for big data processing. *Commun. ACM 59,* 11, 56–65.

ZHU, A. D., MA, H., XIAO, X., LUO, S., TANG, Y., AND ZHOU, S. 2013. Shortest path and distance queries on road networks: towards bridging theory and practice. In *SIGMOD*.

Shuai Ma is a professor at the School of Computer Science and Engineering, Beihang University, China. He received his PhD degrees from University of Edinburgh in 2011 and from Peking University in 2004, respectively. He is a recipient of the VLDB 2010 best paper award and ICDM 2019 best paper candidate. He is an Associate Editor of the VLDB Journal since 2017 and IEEE Transactions on Big Data Since 2020. His current research interests include database theory and systems, and big data.

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 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, and distributed systems.