Big Graphs: Challenges and Opportunities

Wenfei Fan
Shenzhen Institute of Computing Sciences, University of Edinburgh, Beihang University
wenfei@inf.ed.ac.uk

ABSTRACT

Big data is typically characterized with 4V's: Volume, Velocity, Variety and Veracity. When it comes to big graphs, these challenges become even more staggering. Each and every of the 4V's raises new questions, from theory to systems and practice. Is it possible to parallelize sequential graph algorithms and guarantee the correctness of the parallelized computations? Given a computational problem, does there exist a parallel algorithm for it that guarantees to reduce parallel runtime when more machines are used? Is there a systematic method for developing incremental algorithms with effectiveness guarantees in response to frequent updates? Is it possible to write queries across relational databases and semistructured graphs in SQL? Can we unify logic rules and machine learning, to improve the quality of graph-structured data, and deduce associations between entities? This paper aims to incite interest and curiosity in these topics. It raises as many questions as it answers.

PVLDB Reference Format:
Wenfei Fan. Big Graphs: Challenges and Opportunities. PVLDB, 15(12): 3782 - 3797, 2022.
doi:10.14778/3554821.3554899

1 INTRODUCTION

It is increasingly common to find real-life data modeled as graphs, which represent entities as vertices and relationships between entities as edges. Indeed, graphs have found prevalent use in online recommendation, social network analysis, transportation networks, transaction analysis, link prediction, association deduction, event prediction, fraud detection and drug discovery, among other things. Graphs have made an important source of big data.

Big data is typically characterized with 4V's, namely, Volume, Variety, Velocity and Veracity. Already hard for structured relational data, these issues are even more intricate for semistructured graphs. Each and every of these issues introduces new challenges, calls for new techniques, and demands a departure from traditional theory, systems and practice. At the same time, with the new challenges come new opportunities for researchers and practitioners.

To illustrate the challenges and opportunities, for each of the 4V issues, this paper picks and discusses a couple of research topics.

(1) Volume: Parallel computation. Consider a class $\mathcal{Q}$ of graph pattern queries. Given a query $Q \in \mathcal{Q}$ and a graph G, we want to compute the set $Q(G)$ of all matches of pattern Q in graph G. When pattern matching is defined in terms of subgraph isomorphism, it is intractable even to decide whether $Q(G)$ is empty (cf. [83]). In the real world, graphs easily have billions of vertices and trillions of edges, *e.g.,* the social graph at Facebook and the transaction graph at Alibaba Group. It is often prohibitively costly to compute $Q(G)$ in such a graph, even when we define pattern matching in terms of graph simulation [143], which takes quadratic-time [96].

An industrial approach to coping with the volume of big graphs is parallel computation. Several parallel graph systems have been developed, *e.g.,* Pregel [135], PowerGraph [87, 131], Trinity [165], GRACE [187], Giraph++ [178], GraphX [88], and Galois [36, 148].

However, there are at least two questions about the approach.

Vertex-centric vs. graph-centric. Most of the systems adopt vertex-centric models [87, 131, 135]; users need to "think like a vertex" when programming. While a large number of conventional sequential algorithms are already in place, to program with the systems, one has to recast the existing algorithms into vertex-centric ones. The recasting is nontrivial for, *e.g.,* algorithms for graph simulation [96]. Moreover, the systems provides no guarantee on the correctness and even the termination of vertex-centric computations.

We argue for a graph-centric model as an alternative. It simplifies parallel programming from "think like a vertex" to "think like a graph", and from "think parallel" to "think sequential". The idea is to parallelize existing sequential algorithms across a cluster of machines. Under a generic condition, it guarantees that the parallelized computation converges at correct answers as long as the sequential algorithms are correct. For computation problems such as graph simulation, the graph-centric model works better than the vertex-centric ones in both efficiency and ease of programming.

A graph-centric model was proposed by GRAPE [77, 78] (GRAPh Engine). GRAPE has been deployed and extended at Alibaba Group, and supports 90+% of daily graph operations there [56]. It is renamed as GraphScope and is open source at Github [1].

Parallel scalability. The assumption behind the parallel systems is the parallel scalability [118]: the more machines are used, the less the parallel runtime is. Unfortunately, the assumption may not hold. For example, Single-Source Shortest Path (SSSP) is "essentially not scalable with an increasing number of machines" [197]. This is because parallel graph systems typically adopt the shared-nothing architecture. The more machines are used, the heavier the communication cost is incurred. Worse yet, for some computation problems, *e.g.,* graph simulation, no parallelly scalable algorithm exists [75].

Then, what graph computation problems are parallelly scalable, *i.e.,* they admit such algorithms? An interesting observation is that such algorithms exist for the intractable problem of subgraph isomorphism, but not for the quadratic-time problem of subgraph simulation, in contrast to the classic polynomial hierarchy [151].

(2) Velocity: Incrementalization of graph algorithms. Real-life graphs are often frequently changed by small updates. Sup-

This work is licensed under the Creative Commons BY-NC-ND 4.0 International License. Visit https://creativecommons.org/licenses/by-nc-nd/4.0/ to view a copy of this license. For any use beyond those covered by this license, obtain permission by emailing info@vldb.org. Copyright is held by the owner/author(s). Publication rights licensed to the VLDB Endowment.
Proceedings of the VLDB Endowment, Vol. 15, No. 12 ISSN 2150-8097.
doi:10.14778/3554821.3554899

pose that we have computed the matches $Q(G)$ of a pattern Q in a graph G. When G is updated by ΔG, we need to compute matches $Q(G \oplus \Delta G)$ in the updated graph $G \oplus \Delta G$ for, *e.g.,* fraud detection. A batch approach is to recompute $Q(G \oplus \Delta G)$ starting from scratch, which is costly for big G. Another approach is by means of an incremental algorithm $\mathcal{A}_\Delta$ that takes Q, G, $Q(G)$ and ΔG as input, and computes changes ΔO to the old output $Q(G)$ such that $Q(G \oplus \Delta G) = Q(G) \oplus \Delta O$, by minimizing unnecessary recomputation. When ΔG is small, update ΔO to $Q(G)$ is often small as well, and the incremental approach is often more efficient than the batch one.

There are two questions about incremental computations.

Effectiveness measure. What is the criterion for measuring the effectiveness of an incremental algorithm $\mathcal{A}_\Delta$? A traditional characterization is by means of a notion of *boundedness* proposed in [177] and extended to graphs in [74, 155]. It measures the cost of $\mathcal{A}_\Delta$ in $|\mathsf{CHANGED}| = |\Delta G| + |\Delta O|$, the size of the changes in the input and output. Algorithm $\mathcal{A}_\Delta$ is called *bounded* if its cost can be expressed as a function of $|\mathsf{CHANGED}|$ and $|Q|$. The incremental problem for $\mathcal{Q}$ is *bounded* if there exists a bounded $\mathcal{A}_\Delta$ for $\mathcal{Q}$, and *unbounded* otherwise. However, the notion of boundedness is too strong: very few bounded incremental algorithms are known and worse yet, a variety of problems have been proven unbounded [71].

We propose a notion of relative boundedness [58, 71]. It measures the speedup of an incremental algorithm $\mathcal{A}_\Delta$ relative to a batch counterpart $\mathcal{A}$ for its computation problem. A variety of practical incremental algorithms can be shown relatively bounded.

Incrementalization. How can we develop incremental algorithms with effectiveness guarantees? Incremental algorithms are hard to write and analyze. While a large number of batch graph algorithms have been developed, few incremental graph algorithms are yet in place, and even fewer can provably guarantee that they outperform their batch counterparts for small ΔG [71]. These call for systematic methods for developing effective incremental algorithms.

We propose to incrementalize existing batch algorithms [64, 71, 73]. For a query class $\mathcal{Q}$, we pick a batch algorithm $\mathcal{A}$ that has been verified effective after years of practice. We deduce an incremental algorithm $\mathcal{A}_\Delta$ from $\mathcal{A}$, by reusing the original logic and data structures of $\mathcal{A}$ as much as possible, rather than to design $\mathcal{A}_\Delta$ starting from scratch. The users of $\mathcal{A}$ can easily understand how $\mathcal{A}_\Delta$ behaves *w.r.t.* different inputs after practicing $\mathcal{A}$ for years. Moreover, when $\mathcal{A}$ satisfies certain conditions, one can show that the deduced $\mathcal{A}_\Delta$ is both correct and bounded relative to $\mathcal{A}$.

(3) Variety: Queries across relations and graphs. A question raised by our FinTech collaborators asks how they can write queries across a relational database $\mathcal{D}$ and a schemaless graph G, in SQL?

The need for studying this is evident. While business data often resides in relational databases, it is increasingly common to find graph-structured data. With this comes the need for synthesizing data across $\mathcal{D}$ and G, to correlate their information pertaining to the same entities. After all, the added value of big data comes from diverse data sources. Moreover, practitioners often want to write the queries in SQL after practicing SQL for decades.

Nonetheless, with the practical need come two questions.

*Heterogeneous entity resolution (*HER*).* How can we accurately determine whether a tuple t in $\mathcal{D}$ and a vertex v in G refer to the same real world entity? Unlike relational data, real-life graphs may not come with a schema. Even in the same graph, entities of the same "type" may have heterogeneous topological structures, and their properties are often linked via paths, rather than as attributes. While there has been a large body of work on entity resolution (ER), HER across relations and graphs remains unsettled.

We present a notion of parametric simulation for HER [52], which embeds semantic matching into topological matching. It is inductively defined to assess the semantic closeness of "descendants" and decide whether t matches v based on the "global" information.

Semantic joins. How can we support SQL queries across relations $\mathcal{D}$ and graphs G? This requires us to align entities across $\mathcal{D}$ and G, and retrieve data by traversing G. Neither DBMSs nor federated systems (polyglot systems, multistores and polystores) support these yet.

We propose a semantic extension of SQL joins. If a tuple t in $\mathcal{D}$ and v in G are determined by HER to refer to the same real-world entity, then we can naturally "join" the two, extract relevant properties of vertex v and enrich tuple t with the additional "attributes".

(4) Veracity: The quality and values of big graph. Real-life data is often dirty. It is common to find duplicates and semantic inconsistencies even in knowledge graphs widely in use. Indeed, noise in biomedical knowledge graphs is considered a big challenge to drug discovery [202]. Dirty data is costly. It is estimated that poor data quality is responsible for an average of $15 million per year in losses for organizations [84], and costs the US $3.1 trillion in 2016 alone (cf. [157]). Data-driven decisions based on dirty data can be worse than making decisions with no data. With this comes the need for data cleaning, to accurately detect and fix errors in the data.

An immediate question is how to clean semistructured graphs?

Quality of graph data. The veracity is often considered the most challenging issue of the 4V's for big data. Already hard for relational data, it is far more difficult to clean graphs in the absence of a schema. There has been a host of work on relational data cleaning, approached via either machine learning (ML) or logic rules. However, the study of graph cleaning is still in its infancy.

We advocate an approach to cleaning graphs by unifying logic deduction and ML [50, 60]. The idea is to embed ML models in logic rules as predicates. On the one hand, we can plug well-trained ML models in this uniform logical framework to cover cases overlooked by logic rules. On the other hand, we can discover logic conditions for ML predictions to be true and hence, interpret ML predictions.

The value of big graphs. A related question is what values we can get out of big graph analytics by unifying ML and rules? We have implemented such a uniform framework in Fishing Fort [166], an industrial system for graph analytics. Fishing Fort has proven effective in online recommendation, drug discovery, and capacity grading for Lithium-Ion battery manufacturing, among other things.

Organization. Section 2 discusses parallel models and parallel scalability. Section 3 presents effectiveness measures and methods for incrementalizing batch algorithms. Section 4 addresses entity resolution and SQL queries across relations and graphs. Section 5 advocates a combination of machine learning and logic rules for cleaning graphs and getting values out of graphs. Finally, Section 6 identifies open research issues in connection with big graphs.

It should be remarked that the paper aims to incite interest and curiosity in the study of the 4V's of big graphs. It is by no means a comprehensive survey. It raises as many questions as it answers.

2 VOLUME: PARALLEL COMPUTATION

This section targets two issues in connection with parallel graph computation. What parallel computation models should we employ to answer graph queries (Section 2.1)? Would parallel processing suffice to cope with the volume of big graphs (Section 2.2)?

2.1 Parallel Models

The most popular model for parallel graph algorithms is the vertex-centric model, pioneered by Pregel [135] and PowerGraph [87, 131]. The idea is for programmers to "think like a vertex". For instance, to program with Pregel, one needs to write a user-defined function *compute(msgs)* to be executed at a vertex v, where v communicates with other vertices by message passing (*msgs*). This requires the users to recast existing sequential graph algorithms into vertex-centric ones. The recasting is often nontrivial, and makes parallel graph computations a privilege for experienced users only.

As an alternative, we present the graph-centric model of GRAPE [77, 78]. By parallelizing existing sequential graph algorithms, it shifts parallel graph programming from "think parallel" to "think sequential" just like conventional programming.

Consider directed or undirected graphs $G = (V, E, L)$, where V is a finite set of vertices; $E \subseteq V \times L(E) \times V$ is a set of edges (v, l, v') with label l; and each v in V is labeled with its "content" $L(v)$.

PIE programs. Consider a class $\mathcal{Q}$ of graph queries. GRAPE answers queries of $\mathcal{Q}$ via data-partitioned parallelism by executing a PIE program (see below). It works with n workers $P_1, \ldots, P_n$ and a master P_0. It partitions graph G into n fragments $(F_1, \ldots, F_n)$ with an existing partitioner [24, 59, 111], and distributes the fragments to workers such that fragment F_i resides at worker P_i ($i \in [1, n]$).

To develop a parallel algorithm for answering queries of $\mathcal{Q}$ with GRAPE, the user only needs to provide three sequential algorithms, referred to as *a* PIE *program for* $\mathcal{Q}$ (PEval, IncEval and Assemble).

(1) PEval: A *sequential* algorithm that given a query $Q \in \mathcal{Q}$ and a graph G, computes the answer $Q(G)$ to Q in G.
(2) IncEval: A *sequential incremental* algorithm that given $Q, G, Q(G)$ and updates ΔG to G, computes updates ΔO to $Q(G)$ such that $Q(G \oplus \Delta G) = Q(G) \oplus \Delta O$, where $G \oplus \Delta G$ denotes G updated by ΔG.
(3) Assemble: A function that collects partial answers computed locally at each worker by PEval and IncEval, and assembles the partial results into $Q(G)$. This function is typically straightforward.

PEval and IncEval can be any *existing sequential* algorithms for $\mathcal{Q}$. The only additions are the following declarations in PEval.

(a) Update parameters. PEval declares (a) a set C_i of vertices in fragment F_i as the *update region* of F_i; and (b) *status variables* $\bar{x}$ for C_i. We denote by $C_i.\bar{x}$ the set of *update parameters* of F_i, *i.e.,* the status variables associated with the vertices in C_i. Intuitively, $C_i.\bar{x}$ marks candidates to be updated by the incremental steps of IncEval.

(b) Aggregate functions. PEval also specifies an aggregate function f_{aggr}, *e.g.,* min and max, to resolve conflicts when multiple workers attempt to assign different values to the same update parameter.

Fixpoint model. GRAPE parallelizes the execution of a PIE program ρ, which can be modeled as a simultaneous fixpoint operator defined on n fragments. To simplify the discussion, consider the Bulk Synchronous Parallel model (BSP) [182]. The parallelized computation starts with PEval for partial evaluation [105], and conducts incremental computation in supersteps by taking IncEval as the intermediate consequence operator, as follows:

$$R_i^0 = \text{PEval}(Q, F_i^0[\bar{x}_i]),$$
$$R_i^{r+1} = \text{IncEval}(Q, R_i^r, F_i^r[\bar{x}_i], M_i),$$

where $i \in [1, n]$, r denotes a superstep, R_i^r represents the partial results in step r at worker P_i, fragment $F_i^0 = F_i$, $F_i^r[\bar{x}_i]$ is fragment F_i at the end of superstep r bearing update parameters $C_i.\bar{x}$, and M_i is a message carrying changes to update parameters $C_i.\bar{x}$.

More specifically, upon receiving a query $Q \in \mathcal{Q}$ at master P_0, GRAPE posts Q to all workers and computes $Q(G)$ as follows.

*(1) Partial evaluation (*PEval*)*. In the first superstep, GRAPE computes partial results $R_i^0 = \text{PEval}(Q, F_i)$ in fragment F_i at each worker P_i by invoking PEval, in parallel ($i \in [1, n]$). After $Q(F_i)$ is computed, worker P_i sends its set $C_i.\bar{x}$ to master P_0 as a message.

For each status variable $x \in C_i.\bar{x}$, master P_0 collects a multiset S_x of values from messages of all workers. It computes $x_{\text{aggr}} = f_{\text{aggr}}(S_x)$ by applying the aggregate function f_{aggr} declared in PEval, to resolve conflicts. It generates message M_i to worker P_i, which includes only those $f_{\text{aggr}}(S_x)$'s such that $f_{\text{aggr}}(S_x) \neq x$, *i.e.,* only the *changed* values of the update parameters of fragment F_i.

*(2) Incremental computation (*IncEval*)*. In superstep $r + 1$, upon receiving message M_i from master P_0, each worker P_i invokes IncEval to *incrementally* compute $R_i^{r+1} = \text{IncEval}(Q, R_i^r, F_i^r, M_i)$ by *treating message* M_i *as updates* to F_i^r, in parallel ($i \in [1, n]$). It refines its partial results based on the information of M_i from other workers.

At the end of the superstep, P_i sends a message to P_0 that consists of *updated values* of $C_i.\bar{x}$, if any. After receiving messages from all workers, master P_0 deduces a message M_i just like in PEval. It sends message M_i to worker P_i in the next superstep.

*(3) Termination (*Assemble*)*. At each superstep, master P_0 checks whether for all $i \in [1, m]$, P_i is inactive, *i.e.,* P_i is done with its local computation, and there exists no more change to the update parameters of F_i. That is, $R_i^{r_0+1} = R_i^{r_0}$ at a fixpoint r_0 for all $i \in [1, m]$ If so, GRAPE pulls partial results from all workers, and applies Assemble to group them together and get the final result at P_0, denoted by $\rho(Q, G)$. It returns $\rho(Q, G)$ and terminates.

Example 1: We show how GRAPE parallelizes the computation of Single Source Shortest Path (SSSP). Consider a directed graph $G = (V, E, L)$ in which for each edge e, $L(e)$ is a positive number. The length of a path $(v_0, \ldots, v_k)$ in G is the sum of $L(v_{i-1}, v_i)$ for $i \in [1, k]$. For a pair (s, v) of vertices, denote by $\text{dist}(s, v)$ the *distance* from s to v, *i.e.,* the length of a shortest path from s to v. Given graph G and a vertex s in V, SSSP computes $\text{dist}(s, v)$ for all $v \in V$.

The PIE program for SSSP consists of (1) Dijkstra's algorithm for SSSP [80] as PEval, (2) a sequential incremental algorithm of [154] as IncEval, and (3) a straightforward Assemble. We partition graph G via edge cut [24]. We take the set $F_i.O$ of "border nodes" as C_i at each worker P_i, *i.e.,* the vertices in F_i with edges to other fragments. Denote by $F_i.I$ the set of vertices of F_i to which there are edges from other fragments. Let $\mathcal{F}.O = \bigcup_{i \in [1,m]} F_i.O$, and $\mathcal{F}.I = \bigcup_{i \in [1,m]} F_i.I$.

Input: A fragment $F_i(V_i, E_i, L_i)$, and a source vertex s.
Output: $Q(F_i)$ consisting of current $\text{dist}(s, v)$ for all $v \in V_i$.

Declaration: /*candidate set C_i is $F_i.O$*/
For each vertex $v \in V_i$, an integer variable $\text{dist}(s, v)$;
message $M_i := \{\text{dist}(s, v) \mid v \in F_i.O\}$;
aggregate function $f_{\text{aggr}} = \min(\text{dist}(s, v))$;

```
/*sequential algorithm for SSSP (pseudo-code)*/
1.  initialize priority queue Que;
2.  dist(s, s) := 0;
3.  for each v in V_i do if v! = s then dist(s, v) := ∞;
4.  Que.addOrAdjust(s, dist(s, s));
5.  while Que is not empty do
6.    u := Que.pop(); // pop vertex with minimal distance
7.    for each child v of u do // only v that is still in Que
8.      alt := dist(s, u) + L_i(u, v);
9.      if alt < dist(s, v) then
10.       dist(s, v) := alt; Que.addOrAdjust(v, dist(s, v));
11. Q(F_i) := {dist(s, v) | v ∈ V_i};
```

Figure 1: Parallel SSSP: Partial evaluation PEval

(1) PEval. As shown in Fig. 1, PEval is Dijsktra's algorithm [80]. We only need to declare (a) a status integer variable $\text{dist}(s, v)$ for each vertex v, initially ∞ (except $\text{dist}(s, s) = 0$); (b) update parameters as $C_i.\bar{x} = \{\text{dist}(s, v) \mid v \in F_i.O\}$, *i.e.,* the status variables of the border nodes in $F_i.O$ at F_i; and (c) min as an aggregate function f_{aggr}.

At the end of its process, PEval sends $C_i.\bar{x}$ to P_0. Master P_0 maintains $\text{dist}(s, v)$ for all $v \in \mathcal{F}.O = \mathcal{F}.I$. After getting messages from all workers, it takes the smallest value for each $\text{dist}(s, v)$ by applying aggregate function min. It finds those with smaller $\text{dist}(s, v)$ for $v \in F_j.O$, groups them into message M_j, and sends M_j to P_j.

(2) IncEval. As shown in Fig. 2, IncEval is the sequential incremental algorithm for SSSP in [155] that is mildly revised to handle changed $\text{dist}(s, v)$ for v in $F_i.I$ (deduced from $\mathcal{F}.I = \mathcal{F}.O$). Using a queue Que, it starts with changes in M_i, propagates the changes to affected area, and updates the distances (see [155]). The partial result now consists of the revised distances. At the end of the process, it sends to master P_0 the updated values of those status variables in $C_i.\bar{x}$, as in PEval. It applies function min to resolve conflicts.

(3) Assemble. This function simply takes $Q(G) = \bigcup_{i \in [1,n]} Q(F_i)$, the union of the shortest distances of all vertices in all fragments. □

Convergence. The correctness of the fixpoint computation is characterized as follows. Given a class $\mathcal{Q}$ of graph queries, (a) the sequential algorithm PEval for $\mathcal{Q}$ is *correct* if for all queries $Q \in \mathcal{Q}$ and graphs G, it converges at the answer $Q(G)$ to Q in G; (b) the sequential incremental algorithm IncEval for $\mathcal{Q}$ is *correct* if it correctly updates old output $Q(G)$ to $Q(G \oplus M)$, by computing the changes ΔO to $Q(G)$, given changes (messages) M to the update parameters; and (c) Assemble is *correct* for $\mathcal{Q}$ *w.r.t.* partition strategy $\mathcal{P}$ if it correctly computes $Q(G)$ by assembling the partial answers from all workers, when GRAPE with PEval, IncEval and $\mathcal{P}$ terminates.

We say that GRAPE *correctly parallelizes* a PIE program ρ with partition strategy $\mathcal{P}$ if for all $Q \in \mathcal{Q}$ and graphs G, GRAPE guarantees to reach a fixpoint such that $\rho(Q, G) = Q(G)$.

It is shown [70, 78] that GRAPE correctly parallelizes a PIE program ρ for $\mathcal{Q}$ with any partition strategy $\mathcal{P}$ if (a) PEval and IncEval of ρ are correct sequential algorithms for $\mathcal{Q}$, and (b) Assemble correctly combines partial results, and (c) PEval and IncEval satisfy a monotonic condition. The condition is as follows: for all status variables $x \in C_i.\bar{x}$, $i \in [1, m]$, (a) the values of x are from a finite set computed from the active domain of G and (b) there exists a partial order p_x on the values of x such that IncEval updates x in the order of p_x.

Input: A fragment $F_i(V_i, E_i, L_i)$, partial result $Q(F_i)$, and message M_i.
Output: $Q(F_i \oplus M_i)$.

Declaration: message $M_i = \{\text{dist}(s, v) \mid v \in F_i.O,\ \text{dist}(s, v) \text{ decreased}\}$;

```
1. initialize priority queue Que;
2. for each dist(s, v) in M_i do
3.   Que.addOrAdjust(v, dist(s, v));
4. the same as lines 5-11 of in the batch algorithm of Figure 1;
```

Figure 2: Parallel SSSP: Incremental evaluation IncEval

For instance, the PIE program in Example 1 converges at correct $Q(G)$. Updates to $C_i.\bar{x}$ are "monotonic": the value of $\text{dist}(s, v)$ for vertex v is computed from the active domain of G and does not increase. Moreover, $\text{dist}(s, v)$ is the shortest distance from s to v as warranted by the sequential algorithms [80, 155] (PEval and IncEval).

Properties. The graph-centric model has the following properties.

(1) Ease of programming. GRAPE allows users to "plug in" existing sequential algorithms and parallelizes them, without recasting them into a new model or changing the logic of the algorithms. To experience this, one can try to develop a parallel algorithm for graph simulation [96] under the vertex-centric model and the graph centric model [78]. The parallelization makes parallel graph computations accessible to users who are more familiar with conventional graph algorithms. This said, programming with GRAPE still requires users to declare update parameters and design aggregate functions.

(2) Convergence. GRAPE *parallelizes* the computation across a cluster of machines, based on a fixpoint computation with partial evaluation and incremental computation. Under a monotonic condition, the parallelized computation guarantees to converge at correct answers as long as the sequential algorithms provided are correct.

(3) Optimization. GRAPE inherits optimization techniques developed for sequential graph algorithms, *e.g.,* indexing and compression, since it executes sequential algorithms on graph fragments, which are graphs themselves. Moreover, it reduces the costs of iterative graph computations by using IncEval, to minimize unnecessary recomputations. As shown in [56], GRAPE substantially improves the performance of search, cyber security monitoring, ML model training, fraud detection and online recommendation at Alibaba.

(4) Synchronous and asynchronous models. It has been shown that under general conditions, GRAPE also guarantees to converge at correct answers under the Adaptive Asynchronous Model (AAP) [67, 70]. AAP subsumes BSP and asynchronous parallel model (AP) as special cases. It reduces stragglers of BSP and stale computations of AP by learning parameters to measure (a) its progress of a worker relative to other workers, and (b) the staleness of messages.

2.2 Parallel Scalability

Consider a sequential algorithm $\mathcal{A}$ designed for a class $\mathcal{Q}$ of graph queries. Let $t(|Q|, |G|)$ be the worst-case runtime of $\mathcal{A}$ when answering queries Q of $\mathcal{Q}$ in graph G. Following [118], we say that a parallel algorithm $\mathcal{A}_p$ for $\mathcal{Q}$ is *parallelly scalable relative to* $\mathcal{A}$ if for any query $Q \in \mathcal{Q}$ and graph G, the runtime of $\mathcal{A}_p$ for answering Q in G using n machines in parallel can be expressed as:

$$T(|Q|, |G|, n) = O\Big(\frac{t(|Q|, |G|)}{n}\Big).$$

Intuitively, the parallel scalability guarantees speedup of $\mathcal{A}_p$ relative to a "yardstick" sequential $\mathcal{A}$. Such $\mathcal{A}_p$ is able to reduce the cost of $\mathcal{A}$ when more machines are used, and thus scale with large G.

Unfortunately, we cannot take the parallel scalability for granted. For instance, the parallel scalability is beyond reach [75] for graph simulation [143] relative to the quadratic-time algorithm of [96]. This is not surprising. The degree of parallelism is constrained by the *depth* of a computation, *i.e.,* the longest chain of dependencies among its operations [106]. As a consequence, some graph computation problems are "inherently sequential" [89].

On the other hand, parallelly scalable algorithms are known for subgraph isomorphism, *e.g.,* [50, 57, 69], an intractable problem. Taken together with the negative result for graph simulation, these tell us that the parallel scalability does not concur with the polynomial hierarchy [151] in the classic computational complexity theory.

A natural question asks how we should classify graph computation problems *w.r.t.* the parallel scalability under the shared-nothing architecture when we take both the computational cost and communication cost into account? What problems are parallelly scalable? How can we reduce a problem to one that we know to have a parallelly scalable algorithm, along the same lines as our familiar PTIME reduction for NP problems? Does there exist a complete (the hardest) problem in the class of parallelly scalable problems? These questions are not only of theoretical interest, but also practical. Among other things, given a problem, these help us decide whether or not parallel computation for it suffices to scale with large graphs.

3 VELOCITY: INCREMENTALIZATION

This section addresses two questions about incremental graph algorithms. What incremental algorithms are "good" for coping with the velocity of big graphs (Section 3.1)? How can we systematically develop good incremental algorithms (Section 3.2)?

3.1 Relative Boundedness

Ideally, given a class $\mathcal{Q}$ of graph queries, we hope to find a bounded incremental algorithm $\mathcal{A}_\Delta$ for $\mathcal{Q}$ such that its cost can be expressed as a function of the size $|Q|$ of the input query and $|\mathsf{CHANGED}| = |\Delta G| + |\Delta O|$ (the size of changes in the input and output) [155, 177], since $|\mathsf{CHANGED}|$ characterizes the updating cost that is *inherent to* the incremental problem itself. A bounded $\mathcal{A}_\Delta$ warrants efficient incremental computation no matter how big graph G grows.

Unfortunately, bounded incremental algorithms are only in place for the shortest path problems, single-source or all pairs, with positive lengths [155, 156]. Worse still, a variety of incremental problems have been proven unbounded, for which no bounded incremental algorithms exist, *e.g.,* single-source reachability to all vertices (under unit edge deletions) [155], subgraph isomorphism [74], strongly connected components, regular path queries, keyword search and maximum cardinality matching in bipartite graphs [71], even under unit edge deletions/insertions. Moreover, for a bounded incremental problem, it is nontrivial to develop a bounded incremental algorithm, which involves delicate design of auxiliary structures.

Incrementalizing batch algorithms. We promote an alternative approach suggested by our industry collaborators, referred to as *incrementalization.* Given a class $\mathcal{Q}$ of graph queries, it is to deduce an incremental algorithm $\mathcal{A}_\Delta$ for $\mathcal{Q}$ from a popular batch algorithm $\mathcal{A}$ for $\mathcal{Q}$, by reusing the data structures and computation logic of $\mathcal{A}$. The reason is three-fold. (a) A number of *batch algorithms* $\mathcal{A}$ have been developed after decades of study; given a query $Q \in \mathcal{Q}$ and a graph G, $\mathcal{A}$ computes the answers $Q(G)$ to Q in G. It is natural for one to want to incrementalize existing batch ones instead of designing a new one starting from scratch. (b) When practitioners get used to a batch algorithm $\mathcal{A}$ and are familiar with its behaviors in response to different inputs, *e.g.,* after hyper-parameter tuning [46], they often want to stick to $\mathcal{A}$. (c) As will be seen shortly, under certain conditions, it is possible to systematically incrementalize batch algorithms with provable performance guarantees.

Relative boundedness. How can we measure the effectiveness of the incrementalized algorithms? Consider a batch algorithm $\mathcal{A}$ for $\mathcal{Q}$. For a query Q in $\mathcal{Q}$ and a graph G, denote by $G_{(\mathcal{A},Q)}$ the data accessed by $\mathcal{A}$ for computing $Q(G)$, including the auxiliary structure used by $\mathcal{A}$. For updates ΔG to G, denote by AFF the difference between $(G \oplus \Delta G)_{(\mathcal{A},Q)}$ and $G_{(\mathcal{A},Q)}$, *i.e.,* the difference in the data inspected by $\mathcal{A}$ for computing $\mathcal{A}(Q, G \oplus \Delta G)$ and $\mathcal{A}(Q, G)$. We use $|\mathsf{AFF}|$ as a parameter for measuring the cost of $\mathcal{A}_\Delta$.

An incremental algorithm $\mathcal{A}_\Delta$ for $\mathcal{Q}$ is *bounded relative* to $\mathcal{A}$ [71] if for any query Q in $\mathcal{Q}$, graph G and updates ΔG to G, the size of the data checked by $\mathcal{A}_\Delta$ can be expressed as a function of the sizes $|Q|$, $|\Delta G|$ and $|\mathsf{AFF}|$. Here AFF includes changes ΔO to output $Q(G)$.

A incremental problem is *bounded relative to* $\mathcal{A}$ if there exists an incremental $\mathcal{A}_\Delta$ that is bounded relative to $\mathcal{A}$.

Intuitively, $|\mathsf{AFF}|$ indicates the affected area by ΔG relative to $\mathcal{A}$, which is necessarily inspected by batch algorithm $\mathcal{A}$ in response to ΔG. Hence a bounded algorithm $\mathcal{A}_\Delta$ relative to $\mathcal{A}$ incurs only the "necessary" cost for any possible incrementalization of $\mathcal{A}$.

The notion of relative boundedness is weaker than the boundedness of [177]. As a consequence, a variety of problems are bounded relative to popular batch algorithms [71], *e.g.,* SSSP [80], graph simulation [96], depth-first search [176], connectivity [16], local clustering coefficient [192], regular path queries [141] and maximum cardinality matching [99], including unbounded problems.

3.2 Incrementalization of Graph Algorithms

We present the method of [73]. It identifies a class of batch graph algorithms, referred to as fixpoint algorithms. It shows that under a generic condition, from each fixpoint algorithm $\mathcal{A}$, an incremental algorithm $\mathcal{A}_\Delta$ can be deduced such that $\mathcal{A}_\Delta$ is correct and bounded relatively to $\mathcal{A}$; moreover, $\mathcal{A}_\Delta$ adopts the same logic and data structures of $\mathcal{A}$, at most using timestamps as an auxiliary structure.

Fixpoint algorithms. Given a query $Q \in \mathcal{Q}$ and a graph G, a batch algorithm $\mathcal{A}$ often computes $Q(G)$ by adopting the following.

- A set $\Psi_{\mathcal{A}}$ of status variables associated with vertices/edges of G.
- Data structures $D_{\mathcal{A}}$, including status in $\Psi_{\mathcal{A}}$ and auxiliary structure for keeping track of the (partial) results of the computation.
- An *update function* f_{x_i}: for each status variable $x_i \in \Psi_{\mathcal{A}}$, it computes the value of x_i, *i.e.,* $x_i = f_{x_i}(Y_{x_i})$, where $Y_{x_i} \subseteq \Psi_{\mathcal{A}}$.
- A logical statement σ_{x_i} on status variables such that σ_{x_i} is true right after each invocation of $f_{x_i}(Y_{x_i})$. We denote by $\sigma_{\mathcal{A}}$ the conjunction of σ_{x_i} for all x_i's in $\Psi_{\mathcal{A}}$, referred to as *the invariant of* $\mathcal{A}$.

Algorithm $\mathcal{A}$ often operates on G and $D_{\mathcal{A}}$ in rounds, and produces partial results, *i.e.,* values of the variables in $\Psi_{\mathcal{A}}$ in each round.

We say that $\mathcal{A}$ is a *fixpoint algorithm* if it is expressible as

$$(D^{t+1}_{\mathcal{A}},\ H^{t+1}_{\mathcal{A}}) = f_{\mathcal{A}}(D^t_{\mathcal{A}},\ Q,\ G,\ H^t_{\mathcal{A}}), \text{ where}$$

(1) $D^t_{\mathcal{A}}$ denotes the status $D_{\mathcal{A}}$ after $t-1$ rounds of iterations, and $D^0_{\mathcal{A}}$ includes the initial values for all status variables in $\Psi_{\mathcal{A}}$;

(2) $H^t_{\mathcal{A}}$ is a subset of status variables in $\Psi_{\mathcal{A}}$ collected before the start of round t, such that their values are to be necessarily inspected/updated in round t; the smaller $H^t_{\mathcal{A}}$ is, the less costly $\mathcal{A}$ is; we refer to $H^t_{\mathcal{A}}$ as *the scope of round t*; initially, $H^0_{\mathcal{A}}$ contains variables x_i that that violate σ_{x_i} for round 0; and

(3) $f_{\mathcal{A}}$ is the intermediate consequence operator of the fixpoint, called the *step function* of algorithm $\mathcal{A}$. It selects status variables from the scope $H^t_{\mathcal{A}}$ and performs update $f_{x_i}(Y_{x_i})$ on each selected x_i to compute status $D^{t+1}_{\mathcal{A}}$. Moreover, $f_{\mathcal{A}}$ returns the scope $H^{t+1}_{\mathcal{A}}$ that updates $H^t_{\mathcal{A}}$ with *affected* status variables of round t, *i.e.,* those x_i's when the value of some variable in Y_{x_i} is *changed* in round t.

Intuitively, a fixpoint algorithm $\mathcal{A}$ is essentially "update-based". It computes $Q(G)$ by applying its step function $f_{\mathcal{A}}$ in rounds, *guided by* the invariant $\sigma_{\mathcal{A}}$. In round t, by propagating the changes from the last round $t-1$ to the scope $H^t_{\mathcal{A}}$ and corresponding parts of $D^t_{\mathcal{A}}$, $f_{\mathcal{A}}$ identifies the scope $H^{t+1}_{\mathcal{A}}$ for the next round. The process proceeds until it reaches a fixpoint r such that $D^{r+1}_{\mathcal{A}} = D^r_{\mathcal{A}}$ and $H^{r+1} = \emptyset$, *i.e.,* when no more changes can be made. All logical statements in invariant $\sigma_{\mathcal{A}}$ hold when the process terminates.

A variety of graph problems have fixpoint algorithms, *e.g.,* SSSP [80], graph simulation [96], depth-first search [176], connectivity [16], local clustering coefficient [192], and bi-connectivity [176].

Example 2: Dijkstra's algorithm for SSSP [80] is a fixpoint algorithm (see Fig. 1 and Example 1). Its data structure $D_{\mathcal{A}}$ associates each vertex v with a status variable x_v, recording the shortest distance from source s, initialized as ∞ for $v \neq s$ (lines 2-3). It also includes priority queue Que. The scope $H_{\mathcal{A}}$ includes all children of the vertices in Que (line 1). Initially, Que only contains x_s.

Its step function $f_{\mathcal{A}}$ is defined in lines 6-10. Each time $f_{\mathcal{A}}$ pops a vertex v from Que. If logical statement σ_{x_u} does not hold for v's children u (*i.e.,* if $x_u \neq f_{x_u}(Y_{x_u})$), it applies update function f_{x_u} to x_u, setting it to $\min_{x_v \in Y_{x_u}} \{x_v + L(v,u)\}$ (lines 7-10). Here Y_{x_u} includes status variables of u's children. Function $f_{\mathcal{A}}$ also adjusts Que accordingly, and the changes will be propagated to the next round. This is how step function decides the scope $H^{t+1}_{\mathcal{A}}$ for the next round. The process terminates when Que and the scope become empty. At this time, the invariant $\sigma_{\mathcal{A}}$ (shortest distances) holds. □

Incrementalization. Given a fixpoint algorithm $\mathcal{A}$, we deduce an incremental algorithm $\mathcal{A}_\Delta$ from $\mathcal{A}$. Suppose that given a graph G and a query $Q \in \mathcal{Q}$, batch algorithm $\mathcal{A}$ computes $Q(G)$ and ends up with a fixpoint $D^r_{\mathcal{A}}$. Then $\mathcal{A}_\Delta$ starts from $D^r_{\mathcal{A}}$. It additionally takes updates ΔG as input, and possibly extends $D_{\mathcal{A}}$ to $D_{\mathcal{A}_\Delta}$ with timestamps. It employs $H^t_{\mathcal{A}_\Delta}$ and $f_{\mathcal{A}_\Delta}$, which are minor extensions of their counterparts of $\mathcal{A}$ to cope with timestamps.

Along the same lines as $\mathcal{A}$, it iterates in rounds to identify scope $H_{\mathcal{A}_\Delta}$ and compute new status $D_{\mathcal{A}_\Delta}$ as follows:

$$\begin{aligned}(D^0_{\mathcal{A}_\Delta}, H^0_{\mathcal{A}_\Delta}) &= h(D^r_{\mathcal{A}}, \Delta G),\\ (D^{t+1}_{\mathcal{A}_\Delta},\ H^{t+1}_{\mathcal{A}_\Delta}) &= f_{\mathcal{A}_\Delta}(D^t_{\mathcal{A}_\Delta},\ Q,\ G,\ H^t_{\mathcal{A}_\Delta}).\end{aligned}$$

Input: Graph $G = (V, E, L)$, source s, updates ΔG, previous fixpoint $D^r_{\mathcal{A}}$.
Output: The updated shortest distance x_v for each v in $G \oplus \Delta G$.

1. $(D_{\mathcal{A}_\Delta}, H_{\mathcal{A}_\Delta}) \leftarrow h(D^r_{\mathcal{A}}, \Delta G)$; /* *apply initial scope function h* */
2. initialize a priority queue Que;
3. **for each** child v of vertex in $H_{\mathcal{A}_\Delta}$ **do**
4. Que.addOrAdjust(v, x_v);
5. the same lines 5-11 as in the batch SSSP algorithm from Figure 1;

Figure 3: Incrementalized algorithm for SSSP

Here h is an *initial scope function* that identifies scope $H^0_{\mathcal{A}_\Delta}$ for $\mathcal{A}_\Delta$. It is derived from the old fixpoint $D^r_{\mathcal{A}}$ and updates ΔG. It initializes auxiliary structures and changes $D^r_{\mathcal{A}}$ to status $D^0_{\mathcal{A}_\Delta}$.

Incremental algorithm $\mathcal{A}_\Delta$ works along the same lines as batch algorithm $\mathcal{A}$. It starts from $D^0_{\mathcal{A}_\Delta}$ and $H^0_{\mathcal{A}_\Delta}$. Moreover, it employs step function $f_{\mathcal{A}_\Delta}$ to identify scope $H^{t+1}_{\mathcal{A}_\Delta}$ and update status to $D^{t+1}_{\mathcal{A}_\Delta}$ in round t. The process iterates until it reaches a fixpoint.

More specifically, the step function $f_{\mathcal{A}_\Delta}$ (resp. status $D_{\mathcal{A}_\Delta}$) of $\mathcal{A}_\Delta$ extends $f_{\mathcal{A}}$ (resp. $D_{\mathcal{A}}$) of its batch counterpart $\mathcal{A}$ only to cope with newly added timestamps in $S_{\mathcal{A}_\Delta}$. That is, incremental algorithm $\mathcal{A}_\Delta$ essentially adopts the same logic and data structures of $\mathcal{A}$. It differs from $\mathcal{A}$ mostly in the use of initial scope function h.

The initial scope function h determines initial status $D^0_{\mathcal{A}_\Delta}$ and scope $H^0_{\mathcal{A}_\Delta}$ for which the corresponding logical statements are violated by the updates ΔG. For instance, the shortest distance value of some variable x_v may become invalid in SSSP if vertex v's adjacent edges evolve. Function h finds all status variables affected by ΔG, and tunes affected variables to their "feasible" status, from where the new correct result can be computed by resuming $\mathcal{A}$'s iterative computation. An algorithm for deducing such a function h is given in [73], which guarantees $H^0_{\mathcal{A}_\Delta} \subseteq \mathsf{AFF}$, *i.e.,* it checks all and only necessary status variables affected by ΔG.

Performance guarantees. A method for incrementalizing fixpoint algorithms is proposed in [73]. Its main results are as follows.

For any fixpoint algorithm $\mathcal{A}$ for $\mathcal{Q}$, an incremental algorithm $\mathcal{A}_\Delta$ for $\mathcal{Q}$ can be deduced from $\mathcal{A}$ such that $\mathcal{A}_\Delta$ is correct, *i.e.,* given any query $Q \in \mathcal{Q}$, graph G and updates ΔG, $\mathcal{A}_\Delta$ computes ΔO such that $Q(G \oplus \Delta G) = Q(G) \oplus \Delta O$. Moreover, if $\mathcal{A}$ is contracting and monotonic, then $\mathcal{A}_\Delta$ is bounded relative to $\mathcal{A}$.

Here $\mathcal{A}$ is *contracting* if there exists a partial order $\preceq$ such that the status variables in $\Psi_{\mathcal{A}}$ are updated following the partial order. It is *monotonic* if for each status variable $x_i \in \Psi_{\mathcal{A}}$, the update function f_{x_i} is monotonic, *i.e.,* $Y^1_{x_i} \preceq Y^2_{x_i}$ implies that $f_{x_i}(Y^1_{x_i}) \preceq f_{x_i}(Y^2_{x_i})$.

Furthermore, $\mathcal{A}_\Delta$ adopts the same logic and data structure as $\mathcal{A}$; more specifically, (a) the data structure $D_{\mathcal{A}_\Delta}$ extends $D_{\mathcal{A}}$ only by (possibly) associating a timestamp with (some of) its status variables x_i, to record the time of the last change to x_i and identify what changes to status variables have to be propagated; (b) the step function $f_{\mathcal{A}_\Delta}$ is the same as $f_{\mathcal{A}}$ except that it updates the timestamp of x_i when x_i is updated; and (c) scope $H_{\mathcal{A}_\Delta}$ extends $H_{\mathcal{A}}$ similarly.

Example 3: An incrementalization of Dijkstra's algorithm for SSSP [80] is shown in Fig. 3, which employs the initial scope function h deduced in [73], and initializes Que with the updated status identified by h. It adopts the same logic and data structure of the batch algorithm in Fig. 1, without using timestamps. Except the initialization, it is the same as the incremental algorithm in Fig. 2. □

Remark. (1) There are graph algorithms that are not expressible as fixpoint, *e.g.,* METIS [107] for graph partitioning. Nonetheless, incremental algorithms can still be deduced from such algorithms and perform well in both the scalability and partition quality [64].

(2) There have been other incrementalization approaches at the instruction level [7, 26, 129, 140] or for vertex-centric graph algorithms via memorization [27, 194, 201] or dependency-driven streaming frameworks [137, 185]. In contrast, (a) we target graph-centric algorithms; (b) we deduce incremental graph algorithm $\mathcal{A}_\Delta$ by reusing the same logic and data structures of its batch counterpart $\mathcal{A}$, in contrast to the instruction-level approach [7, 26, 129]; and (c) under the monotonic and contraction conditions, our incrementalized algorithms guarantee to be correct and relatively bounded.

4 VARIETY: RELATIONS AND GRAPHS

This section tackles two questions in connection with the variety. How can we decide whether a tuple t in a relational database $\mathcal{D}$ and a vertex v in a semistructured graph G refer to the same entity (Section 4.1)? Can we write queries across $\mathcal{D}$ and G in SQL (Section 4.2)?

4.1 Heterogeneous Entity Resolution

To synthesize information across $\mathcal{D}$ and G, effective methods have to be in place for Heterogeneous Entity Resolution (HER), to determine whether a tuple t in $\mathcal{D}$ and a vertex v in G match. The need for this is evident for querying $\mathcal{D}$ and G taken together, integrating data from $\mathcal{D}$ and G, and enriching $\mathcal{D}$ with semantic information from a knowledge graph G, among other things.

Entity resolution (ER) has been well studied for relations [12, 13, 15, 20, 28, 31, 35, 45, 51, 68, 72, 81, 91, 100, 108, 113, 117, 126, 145, 152, 175, 193, 195, 205, 208] and graphs [43, 49, 101, 102, 119, 120, 124, 161, 169, 179, 191, 196, 200, 206, 212]. However, much less is known about HER across relations $\mathcal{D}$ and graph G. To this end, JedAI [150] considers various data formats such as RDF and CSV, by first converting entities to a set of profiles (name-value pairs), and then checking labels and attributes as in [147]. PathSim [171] extends SimRank under a meta path framework to measure similarity via topological matching. MAGNN [82] combines graph neural network with meta-paths to extract embeddings and measure vertex similarity. A model was trained in [207] to link entities in Web tables and knowledge bases. Models were also trained to map cells (attribute values) in a relation to entities in knowledge bases [149, 180].

Unfortunately, the prior methods do not work well on HER across relations $\mathcal{D}$ and graphs G. Relational ER methods rely on schema information, and do not apply to schema-agnostic graphs. In particular, entities are often represented as vertices v in G, and their properties are linked from v via paths. To cope with these, one has to use joins to traverse paths and incur costs way beyond quadratic time (the worst-case complexity of relational ER). Moreover, prior methods explore only local properties, *e.g.,* "local embedding" [28, 196] collects local information of neighbors within limited hops. However, to identify a tuple t in $\mathcal{D}$ and a vertex v in G, one often has to recursively check the pairwise semantic closeness of descendants (key features) of t and v. Cell matching [149, 180] overlooks correlated attributes of tuple t when mapping to vertex v.

We present the method of [52] for HER across relations $\mathcal{D}$ and graph G. It makes an effort to improve the accuracy by embedding semantic matching (ML) into topological matching, and employing inductive matching to collect global information. Moreover, it takes quadratic time in the worst case, the same as for relational ER.

Preliminaries. We start with basic notations. Consider a database schema $\mathcal{R} = (R_1, \ldots, R_n)$, where R_i is a relation schema $(A_1, \ldots, A_k)$, and A_i is an attribute. A *relation of schema* R is a set of tuples with the attributes A_i of R ($i \in [1, k]$). A *database* $\mathcal{D}$ *of* $\mathcal{R}$ is $(D_1, \ldots, D_n)$, where D_i is a relation of R_i ($i \in [1, n]$).

A *path* ξ *from a vertex* v_0 in a graph G is a sequence $\xi = (v_0, v_1, \ldots, v_l)$ such that (v_{i-1}, l_{i-1}, v_i) is an edge in G for $i \in [1, l]$. The *length of* ξ, denoted by $\mathsf{len}(\xi)$, is l, *i.e.,* the number of edges on ξ. A path is *simple* if $v_i \neq v_j$ for $i \neq j$, *i.e.,* a vertex appears on ξ at most once. We consider simple paths in the sequel.

We refer to v_2 as a *child* of v_1 if (v_1, l, v_2) is an edge in E for some label l, and as a *descendant* if there exists a path from v_1 to v_2. A vertex is a called *leaf* if it has no children.

HER**: Overview**. Given a database $\mathcal{D}$ and a graph G, HER first converts $\mathcal{D}$ to a canonical graph G_D offline by, *e.g.,* direct mapping of RDB2RDF [186], which yields an 1-1 mapping f_D from the tuples and their attributes in $\mathcal{D}$ to the vertices and their edges in G_D, respectively. One may use other converting methods (see [142] for a survey). We take RDB2RDF [186] here to simplify the presentation.

HER then learns score functions and bounds offline, for assessing semantic closeness. To determine whether a vertex u_0 in a graph G_1 matches a vertex v_0 in another graph G_2, we inductively considers the "closeness" of descendants of u_0 and descendants of v_0. HER adopts score functions h_v and h_ξ defined as follows:

$$h_v(u', v') = \mathcal{M}_v(L_1(u'), L_2(v'))$$
$$h_\xi(\xi_1, \xi_2) = \frac{\mathcal{M}_\xi(L_1(\xi_1), L_2(\xi_2))}{\mathsf{len}(\xi_1) + \mathsf{len}(\xi_2)}$$

Here $\mathcal{M}_v$ is a function that assesses how close u' and v' are to each other, based on their labels (types and values), and $\mathcal{M}_\xi$ inspects how close the association of u' to u_0 and the association of v' to v_0 are, based on the labels on paths ξ_1 and ξ_2, where ξ_1 (resp. ξ_2) is a path from u_0 to u' (resp. v_0 to v'). Intuitively, the longer a path is, the weaker the association is; hence $\mathcal{M}_\xi(\xi_1, \xi_2)$ is divided by $\mathsf{len}(\xi_1) + \mathsf{len}(\xi_2)$. Both $h_v(u', v')$ and $h_\xi(\xi_1, \xi_2)$ are in [0, 1].

To identify u_0 and v_0 in practice, it often suffices to inspect a small number of their characteristic features (descendants). In light of this, we adopt an ML-based ranking function $h_r(\cdot, \cdot)$ and a bound k such that given a vertex u, $h_r(u, k)$ ranks the descendants of u and selects top-k ones along with a path for each; similarly for $h_r(v, k)$. Denote by V_v^k the set of top-k descendants of v picked by $h_r(v, k)$.

We use $h_r(\cdot, \cdot)$ to strike a balance between the complexity and accuracy. Since there are exponentially many paths to descendants of u, it is impractical to enumerate them when G_1 or G_2 is dense.

After the models are trained, HER conducts matching online. Given a pair (t, v_g) for a tuple t in $\mathcal{D}$ and a vertex v_g in G as input, it finds the vertex u_t in the canonical graph G_D denoting t, via mapping f_D. It then checks whether (u_t, v_g) makes a match via parametric simulation. It returns true if so, and false otherwise.

Parametric simulation. This notion is an extension of graph simulation [143]. It is inductively defined to conduct global checking following [143]. In contrast to [143], it is parameterized with score

functions and closeness thresholds learned via ML models. Moreover, it may map paths in one graph to paths in another. It does not require every edge of u to find a match in G, so as to cope with semistructured graphs in which missing links are common.

Taking functions (h_v, h_ξ, h_r) and thresholds (σ, δ, k) as parameters, *parametric simulation* is to check whether (u_0, v_0) is a match, for u_0 in G_1 and v_0 in G_2 across two graphs $G_1 = (V_1, E_1, L_1)$ and $G_2 = (V_2, E_2, L_2)$. Specifically, given (u_0, v_0), it computes a binary relation $\Pi(u_0, v_0) \subseteq V_1 \times V_2$ satisfying the following conditions:

(1) $(u_0, v_0) \in \Pi(u_0, v_0)$; and
(2) for each pair $(u, v) \in \Pi(u_0, v_0)$,
  (a) $h_v(u, v) \geq \sigma$; and
  (b) if u is a non-leaf, then there is a set $S_{(u,v)}$ of (u', v') that is a partial 1-to-1 mapping from V_u^k to V_v^k such that its aggregate $\sum_{(u',v') \in S_{(u,v)}} h_\xi(\xi_{(u,u')}, \xi_{(v,v')}) \geq \delta$;
  and for each $(u', v') \in S_{(u,v)}$, $(u', v') \in \Pi(u_0, v_0)$.

Here $\xi_{(u,u')}$ is the path from u to u' selected by $h_r(u, k)$; similarly for $\xi_{(v,v')}$. We refer to $S_{(u,v)}$ as a *lineage set* of (u, v).

We say that (u_0, v_0) is a *match* by simulation parameterized with $(h_v, h_\xi, h_r, \sigma, \delta, k)$ if there exists such a nonempty $\Pi(u_0, v_0)$. There are possibly many such sets; to check whether (u_0, v_0) makes a match, it suffices to check the existence of such a set, *i.e.,* a *witness.*

Intuitively, (u_0, v_0) is a match if (1) u_0 and v_0 are close enough, measured by function h_v based on their types and values; (2) there exists a lineage set $S_{(u_0,v_0)}$ of pairwisely matching feature pairs such that their associations to (u_0, v_0) are close enough, measured by the aggregate score with function h_ξ; and (3) for a pair (u, v), $S_{(u,v)}$ is a set of pairs (u', v') such that each characteristic feature u' of u finds the "best" match v' if it exists (hence a partial 1-to-1 mapping) in terms of h_ξ scores on paths found by h_r. That is, (u_0, v_0) is a match if their "values" and key features are close enough.

Remark. (1) It is shown [52] that on average, parametric simulation has F-measure above 0.94 over a variety of relations and graphs.

(2) For any $u_0 \in G_1$ and $v_0 \in G_2$, there exists a unique maximum $\Pi(u_0, v_0)$ by simulation with $(h_v, h_\xi, h_s, \sigma, \delta, k)$ [52]. That is, parametric simulation retains the uniqueness of graph simulation [143].

Implementation. As shown in [52], the vertex model $\mathcal{M}_v$ can be implemented with a sentence embedding model [158]. Using embedding model BERT [40], metric learning model $\mathcal{M}_\xi$ is trained to cope with edge labels. The ranking function h_r first selects a set of m paths from v by using a language model, where m is the number of the children of v; then it ranks the m paths by using a path resource allocation (PRA) algorithm, and returns the top-k ones.

We select σ, δ and k to maximize the F-measure (accuracy). These thresholds are chosen via random search [19] for efficiency.

Once the training is done, it takes linear time for h_v and h_ξ to measure the similarity. It takes $O(|V||E|)$ time for h_r to select top-k features and associated paths for a vertex v in a graph G=(V, E, L).

For a graph G and the canonical graph G_D of database $\mathcal{D}$, it takes $O((|V_D|+|E_D|)(|V|+|E|))$ time for parametric simulation to decide whether a tuple t in $\mathcal{D}$ and a vertex v in G make a match. Such an algorithm is provided in [52], no more expensive than relational ER although it has to traverse G. It is parallelized to compute matches for a set of tuples in $\mathcal{D}$, with the same worst-case complexity.

4.2 Semantic Joins

To determine whether to recommend an investment plan fp_0 to a customer Bob, our FinTech collaborators want to check (a) whether Bob has good credit, and (b) whether there exists a customer, say Ada, who has invested in fp_0 and bought two financial products together with Bob before. This has to check a relational database $\mathcal{D}$ for condition (a), and a transaction graph G for condition (b). Our collaborators want to do it with an SQL query across $\mathcal{D}$ and G.

Employing HER, we propose an approach to answering the query. Consider a database $\mathcal{D}$ of schema $\mathcal{R}$ and a schemaless graph G.

Semantic joins. We develop two functions f and h.

HER *matches*. Given a graph G and a set S of tuples, it computes:

$$f(S, G) = \{(t.\mathrm{id}, v.\mathrm{id}) \mid t \in S, v \in V \text{ in } G, t \Rightarrow v\}.$$

Here $t \Rightarrow v$ denotes that tuple t and vertex v make a *match* decided by HER, *i.e.,* they refer to the same entity. It returns a relation of $(t.\mathrm{id}, v.\mathrm{id})$, where $t.\mathrm{id}$ (resp. $v.\mathrm{id}$) denotes t (resp. v) with identity id.

Attribute Extraction. Given a set S of tuples and a set $\mathcal{A}$ of keywords that indicate users' (query) interest, function h deduces (a) a relation schema $R_G = (\mathrm{vid}, A_1, \ldots, A_m)$, where vid denotes a vertex v, and A_i is an attribute (named by a keyword in $\mathcal{A}$), and (b) an instance $h(S, G)$ of schema R_G by extracting selected properties of the vertices in $f(S, G)$ that match the tuples in S by HER.

Semantic joins. A natural semantic extension to the join operator of relational algebra (RA) is of the form $S \bowtie_{\mathcal{A}} G$, where G is a graph; S denotes a set of tuples of schema R that encode entities; in relational algebra, S is either a relation schema R or a sub-query Q; and $\mathcal{A}$ is a set of keywords that are provided by users to express their interest and specify an *extracted schema* $R_G(\mathrm{vid}, \mathcal{A})$. Essentially, $S \bowtie_{\mathcal{A}} G$ is $S \bowtie f(S, G) \bowtie h(S, G)$ via the SQL join operator. It returns a relation of schema consisting of attributes $\mathrm{attr}(R)$, vid and $\mathcal{A}$, where $\mathrm{attr}(R)$ is the set of attributes of the schema R of S.

Intuitively, for each tuple t in S, $f(S, G)$ identifies vertices v in G such that $t \Rightarrow v$, and $h(S, G)$ enriches t with additional attributes $\bar{A}$ of v extracted from graph G. Since t and v refer to the same entity, $S \bowtie_{\mathcal{A}} G$ correlates their information, extracts properties $\bar{A}$ of v and extends t with $\bar{A}$. We refer to $S \bowtie_{\mathcal{A}} G$ as a *semantic join.*

Implementation. Implementing functions f and h as SQL UDFs, stored procedures, PL/SQL, or a combination of SQL and external scripts, we can convert semantic join $S \bowtie_{\mathcal{A}} G$ to an equivalent SQL query $S \bowtie f(S, G) \bowtie h(S, G)$. Thus $S \bowtie_{\mathcal{A}} G$ is just a syntactic sugar.

As an example, for the FinTech query described earlier, one can use $S \bowtie_{\mathcal{A}} G$ to extract data from the transaction graph G and check condition (b), by taking the set of customers who have invested in fp_0 as S. One can write the query in SQL with a semantic join.

We have seen how to compute HER matches $f(S, G)$ (Section 4.1). We can implement $h(S, G)$ as follows: we (a) train an ML model to select paths from G via, *e.g.,* Long Short-Term Memory (LSTM) networks [45]; (b) group the paths via vectorization and clustering; and (c) select top-ranked clusters that are semantically close to one of the keywords in $\mathcal{A}$, are diverse from the existing attributes of S, and moreover, yield the minimum number of null values. We take these clusters as attributes $\mathcal{A}$ in schema R_G and populate the relation $h(S, G)$ of R_G by following the corresponding paths.

Remark. (1) To simplify the discussion, we have only discussed semantic join $S \bowtie_{\mathcal{A}} G$ to enrich relations. Moreover, one can support link joins to extract the reachability and shortest distances between vertices that match relational tuples. Note that attribute extraction and connectivity require to traverse graph G, and cannot be expressed in first-order logic and relational algebra [127].

(2) We can implement semantic joins on top of database systems and enrich the commercial systems with a capacity of querying relations and graphs. It also sheds lights on data lakes [146] for (a) *query-driven data discovery* to find relevant graphs with vertices matching tuples in $Q(\mathcal{D})$ of a query Q; (b) *on-demand data integration* to augment tuples in $Q(\mathcal{D})$ with properties of matching vertices; and (c) *data extraction* to abstract schema/relations from raw data in graphs.

(3) HER and attribute/link extraction can be plugged into federated systems, for aligning entities across $\mathcal{D}$ and G, and for correlating and synthesizing data about the same entity, respectively. Existing polyglot systems do not support these facilities [4, 37, 39, 48, 103, 104, 153, 168, 210]. Multistores [6, 109, 214] and polystores [44, 92, 112] do not yet support graph storage and computations.

5 VERACITY: ML VS. LOGIC DEDUCTION

This section addresses two questions in connection with the veracity. How can we clean graph-structured data (Section 5.1)? What values can we get out of big graphs (Section 5.2)? We advocate a simple approach to unifying machine leaning and logic deduction.

5.1 The Quality of Graph Data

Graph data quality has two primitive issues: *entity resolution (ER)*, to identify vertices that refer to the same real-world entity; and *conflict resolution (CR)*, to resolve (semantic) inconsistencies of entities.

For relational data, ER and CR have been approached via either machine learning (ML) [11, 12, 35, 42, 95, 114, 134, 145, 152, 159, 173, 184, 205], or logic rules [13, 14, 20, 33, 51, 54, 55, 86, 110, 193]. When it comes to graphs, ML models have been studied for ER [21, 97, 124, 161, 179, 203] via unsupervised clustering and deep learning. Rules for ER and CR include keys for graphs [49], graph functional dependencies (GFDs) [76], graph entity dependencies (GEDs) [66], and numeric graph functional dependencies (NGFDs) [65].

Neither ML models nor logic rules consistently outperform the other. Well-trained ML models are able to cover various cases; but ML predictions are probabilistic and hard to explain. In contrast, logic rules can be interpreted and can fix errors with certainty. However, it is hard to find high-quality rules and cover all the cases.

ML models as predicates. Can we combine ML models and logic deduction, for the two to benefit from each other? Below we present the attempt of [60], which proposes a uniform logical framework based on rules in which ML models can be embedded as predicates.

Graph pattern matching. We start with basic notations.

Graphs. We model a graph as $G = (V, E, L, F_A)$ for the ease of discussion, where V, E and L are the same as given in Section 2.1, and each vertex $v \in V$ carries a tuple $F_A(v) = (A_1 = a_1, \ldots, A_n = a_n)$ of *attributes* of a finite arity, where $A_i \neq A_j$ if $i \neq j$, representing properties. We write $v.A_i = a_i$, where a_i is a constant. We assume a special attribute id at each vertex v, denoting its identity. Different vertices may carry different attributes, not constrained by a schema.

Patterns. We will define rules over graph patterns. A *graph pattern* is $Q[\bar{x}] = (V_Q, E_Q, L_Q, \mu)$, where (1) V_Q (resp. E_Q) is a set of pattern vertices (resp. edges), (2) L_Q assigns a label $L_Q(u)$ (resp. $L_Q(e)$) to each pattern vertex $u \in V_Q$ (resp. edge $e \in E_Q$), and (3) $\bar{x}$ is a list of distinct variables, and μ is a bijective mapping from $\bar{x}$ to V_Q, *i.e.,* it assigns a distinct variable to each vertex v in V_Q. For $x \in \bar{x}$, we use $\mu(x)$ and x interchangeably when it is clear in the context.

Pattern matching. A *match* of pattern $Q[\bar{x}]$ in graph G is a homomorphism h from Q to G such that (a) for each $u \in V_Q$, $L_Q(u) = L(h(u))$; and (b) for each $e = (u, l, u')$ in Q, $e' = (h(u), l, h(u'))$ is an edge in G.

We denote the match as a vector $h(\bar{x})$, consisting of $h(x)$ for all $x \in \bar{x}$ in the same order as $\bar{x}$, as a list of entities identified by Q.

*Graph association rules (*GARs*)*. Below we define the rules.

Predicates. A *predicate* of pattern $Q[\bar{x}]$ is one of the following:

$$p ::= l(x, y) \mid x.A \otimes y.B \mid x.A \otimes c \mid \mathcal{M}(x.\bar{A}, y.\bar{B}),$$

where $\otimes$ is one of $=, \neq, <, \leq, >, \geq$; x and y are variables in $\bar{x}$; c is a constant; A and B are attributes; and $x.\bar{A}$ is a list of attributes at "vertex" x; similarly for $y.\bar{B}$. The predicates are classified as follows.

- Logic predicates: *link predicate* $l(x, y)$ indicates the existence of an edge labeled l from vertex x to y; *variable predicate* $x.A \otimes y.B$ and *constant predicate* $x.A \otimes c$ check the consistency of values.
- ML predicates: $\mathcal{M}(x.\bar{A}, y.\bar{B})$ is an ML classifier that returns true iff $\mathcal{M}$ predicts true at $(x.\bar{A}, y.\bar{B})$. Here $\mathcal{M}$ can be any ML model that returns Boolean (*e.g.,* $\mathcal{M} \geq \sigma$ for a predefined bound σ).

GARs. A *graph association rule* (GAR) φ is defined as

$$Q[\bar{x}](X \rightarrow p_0),$$

where $Q[\bar{x}]$ is a graph pattern, X is a (possibly empty) conjunction of predicates of $Q[\bar{x}]$, and p_0 is a predicate of $Q[\bar{x}]$. We refer to $Q[\bar{x}]$ and $X \rightarrow p_0$ as the *pattern* and *dependency* of φ, respectively.

Intuitively, the pattern Q in a GAR identifies entities in a graph, and the dependency $X \rightarrow p_0$ is applied to the entities. Constant and variable predicates $x.A = c$ and $x.A = y.B$ specify *value associations of* attributes, and link predicates $l(x, y)$ makes *link associations.* Moreover, one can "plug in" pre-trained ML models $\mathcal{M}$ for ER [124], link predictions [188] and similarity checking [40].

The embedded ML predicates allow us to make use of pre-trained ML models and moreover, interpret ML predictions in logic.

Example 4: Below are some GARs over patterns Q_1-Q_4 of Fig. 4.

(1) φ_1=$Q_1[\bar{x}](\mathcal{M}_s(x_1.\text{abstract}, x_2.\text{abstract}) \wedge x_1.\text{seatingCapacity} = x_2.\text{seatingCapacity} \rightarrow x_1.\text{id} = x_2.\text{id})$ is a GAR mined from the knowledge graph DBpedia [123], where $\mathcal{M}_s$ is a model for checking sentence similarity. This GAR can be used for ER; it states that two stadiums x_1 and x_2 can be identified if they have similar abstracts and the same seating capacity, and moreover, they are designed by the same architect and have a common tenant (specified in Q_1).

(2) $\varphi_2 = Q_2[\bar{x}](x_2.\text{genre} = x_3.\text{genre} \wedge \mathcal{M}_e(x_5, x_6) \wedge \mathcal{M}_l(x_2, x_3) \rightarrow x_2.\text{releaseYear} \leq x_3.\text{releaseYear})$ is another GAR from DBpedia for CR. Here $\mathcal{M}_e$ is an ML model for ER and $\mathcal{M}_l$ is a link prediction model that predicts an edge labeled followedBy from x_2 to x_3. This rule says that if two books have the same genre, author and publisher (in Q_2), and if they are from the same series (determined by $\mathcal{M}_e$), then the preceding one x_2 predicted by $\mathcal{M}_l$ is released earlier.

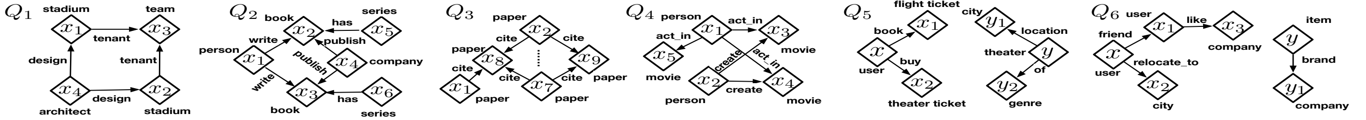

Figure 4: Example graph patterns Q_1–Q_6

(3) $\varphi_3 = Q_3[\bar{x}](\bigwedge_{i,j\in[1,7],i\neq j}(x_i.\text{id} \neq x_j.\text{id} \wedge x_i.\text{venue} = x_j.\text{venue} \wedge \mathcal{M}_s(x_i.\text{topic}, x_j.\text{topic})) \rightarrow \text{cite}(x_1, x_9))$ is from the citation network DBLP [2]. It can predict the link that paper x_1 cites x_9 if there are 7 different papers x_1–x_7 published in the same venue and having similar topics (by $\mathcal{M}_s$), and all of them cite both x_8 and x_9 except x_1.

(4) $\varphi_4 = Q_4[\bar{x}](\bigwedge_{i,j\in[3,5],i\neq j}(x_i.\text{id} \neq x_j.\text{id} \wedge \mathcal{M}_s(x_i.\text{topic}, x_j.\text{topic}) \wedge x_i.\text{language} = x_j.\text{language}) \rightarrow \mathcal{M}_l(x_2, x_5))$ is from the movie database IMDB [3]. It helps interpret the model $\mathcal{M}_l$ which predicts that x_2 is the creator of movie x_5, *i.e.,* this prediction is made because there are 3 different movies x_3–x_5 with similar topics, the same language and a common cast, and x_2 creates x_3 and x_4. □

<u>*Semantics*</u>. Consider a GAR $\varphi = Q[\bar{x}](X \rightarrow p_0)$. Denote by $h(\bar{x})$ a match of Q in a graph G, and by p a predicate of $Q[\bar{x}]$. We write $h(\mu(x))$ as $h(x)$, where μ is the mapping in Q from $\bar{x}$ to vertices in Q.

A match $h(\bar{x})$ *satisfies* a predicate p, denoted by $h(\bar{x}) \models p$, if one of following conditions is satisfied: (a) when p is $l(x, y)$, there exists an edge with label l from $h(x)$ to $h(y)$; (b) when p is $x.A \otimes y.B$, the vertex $h(x)$ (resp. $h(y)$) carries attribute A (resp. B), and $h(x).A \otimes h(y).B$; similarly for constant predicate $h(x).A \otimes c$; and (c) when p is $\mathcal{M}(x.\bar{A}, y.\bar{B})$, the ML model $\mathcal{M}$ predicts true at $(h(x).\bar{A}, h(y).\bar{B})$.

We write $h(\bar{x}) \models X$ if $h(\bar{x}) \models p$ for *all* p in a set X of predicates, We write $h(\bar{x}) \models X \rightarrow p_0$ if whenever $h(\bar{x}) \models X$, then $h(\bar{x}) \models p_0$.

We say that a graph G *satisfies* GAR $\varphi = Q[\bar{x}](X \rightarrow p_0)$, denoted by $G \models \varphi$, if for all matches $h(\bar{x})$ of $Q[\bar{x}]$ in G, $h(\bar{x}) \models X \rightarrow p_0$. We say that G *satisfies* a set Σ of GARs, denoted by $G \models \Sigma$, if for all GARs $\varphi \in \Sigma$, $G \models \varphi$, *i.e., G* satisfies every GAR in Σ.

<u>*Remark*</u>. GARs support all the primitives of relational data cleaning rules. GARs support constant patterns of conditional functional dependencies (CFDs) [54] via $x.A = c$, comparison predicates $=, \neq, <, \leq, >, \geq$ of denial constraints (DCs) [14], and similarity checking of matching dependencies (MDs) [51] via ML models $\mathcal{M}$.

As shown in Example 4, GARs can serve as rules for conducting ER and CR in graphs. GARs subsume graph dependencies GFDs and GEDs as special cases. Besides, GARs may embed ML models as predicates, and moreover, catch missing links with link predicates.

Complexity. There are three classical problems for dependencies.

The *satisfiability* problem is to decide, given a set Σ of GARs, whether there exists a graph G such that $G \models \Sigma$ and for each GAR $Q[\bar{x}](X \rightarrow p_0) \in \Sigma$, Q has a match in G? Intuitively, this is to ensure that all GARs can be applied to G at the same time without conflicts.

A set Σ of GARs *implies* a GAR φ, denoted by $\Sigma \models \varphi$, if for all graphs G, if $G \models \Sigma$ then $G \models \varphi$, *i.e.,* φ is a logical consequence of Σ.

The *implication problem* is to decide, given a set Σ of GARs and a GAR φ, whether $\Sigma \models \varphi$? Intuitively, this is to remove redundant GARs in rule discovery and speed up graph cleaning with GARs.

The *validation problem* is to decide, given a graph G and a set Σ of GARs, whether $G \models \Sigma$? Intuitively, this is to settle the complexity of cleaning graphs by taking GARs as data cleaning rules.

It has been shown that the satisfiability, implication and validation is coNP-complete, NP-complete and coNP-complete, respectively [60, 66]. Here we assume that given two lists of attributes $x.\bar{A}$ and $y.\bar{B}$, checking $\mathcal{M}(x.\bar{A}, y.\bar{B})$ is in PTIME in the sizes $|x.\bar{A}|$ and $|y.\bar{B}|$, as commonly found in practice for pre-trained ML models $\mathcal{M}$.

The complexity bounds are the same as for reasoning about GEDs [66]. The implication and satisfiability analyses are no harder than for relational CFDs, which are also intractable [54].

Algorithms for graph cleaning. Practical algorithms are already in place for discovering GARs from real-life graphs, and for detecting and fixing errors in large-scale graphs. These include (a) algorithms for discovering GARs [50, 61]; (b) parallel algorithms for detecting errors, batch and incremental [60]; and (c) parallel algorithms for fixing errors [69] (the algorithms were developed for GEDs but can be readily extended to GARs). In particular, the algorithms of [50, 60, 69] are parallelly scalable. The algorithms of [69] guarantee that the fixes generated are logical consequences of GARs and ground truth accumulated, *i.e.,* they guarantee to correct errors as long as the GARs and the ground truth are correct.

5.2 The Value of Big Graphs

Besides graph cleaning, what other applications may benefit from the uniform framework of ML predictions and logic deduction? The answer to the question is encouraging. GARs and their variants have been deployed at Fishing Fort [166] and proven effective in a variety of real-life applications. Below we report three cases.

(1) Online recommendation. ML models have been widely used in e-commerce to recommend items to users [94]. The models are often classified as collaborative filtering (CF) and content-based (CB). CF identifies user preference and makes recommendation by learning from user-item historical interactions, *e.g.,* users' previous ratings and browsing history [18, 93, 115, 116, 128, 132, 170, 189, 199]. CB primarily compares the contents of users and items such as user profiles and item features [22, 130, 138, 163, 183, 190, 204]. However, a single strategy, either CF or CB, often does not suffice in practice. For example, instead of exploring new interesting items, CF tends to find similar ones *w.r.t.* the user's past interaction due to its collaborative nature. It does not work well when the interaction data is sparse and when a recommender system starts cold.

To rectify these limitations, hybrid models have been explored to unify interaction-level similarity and content-level similarity, *e.g.,* [9, 29, 30, 32, 172, 181, 213, 216]. However, the hybrid approach often requires to train a new ML model starting from scratch, and it does not explain what is needed to improve a CF or CB model.

Fishing Fort adopts a variant of GARs as an alternative approach. Instead of training a new model, it enriches existing CF, CB and hybrid model $\mathcal{M}$ with additional logic conditions, to reduce both false positives (FPs) and false negatives (FNs) of $\mathcal{M}$. Suppose that $\mathcal{M}$ sets a strength threshold δ such that it recommends item y to user x if $\mathcal{M}(x, y) \geq \delta$. Fishing Fort learns rules of the following forms.

(1) $Q[\bar{x}](\mathcal{M}(x, y) \geq \delta \wedge X_1 \rightarrow (x, \text{likes}, y))$, where $\mathcal{M}$ is an existing recommendation model, and X_1 consists of of logic predicates. Intuitively, while $\mathcal{M}(x, y)$ suggests to recommend item y to user x (*i.e.,* $\mathcal{M}(x, y) \geq \delta$), additional conditions X_1 are checked to filter FPs. That is, item y is recommended to user x only if X_1 holds.

(2) $Q[\bar{x}](\mathcal{M}(x, y) < \delta \wedge X_2 \rightarrow (x, \text{likes}, y))$ to reduce FNs. That is, although $\mathcal{M}(x, y)$ predicts that user x may not like item y (below threshold δ), if logic condition X_2 holds, then y is recommended to x.

As examples, below are two such rules over $Q_5 - Q_6$ of Fig. 4.

(a) $\varphi_5 = Q_5[\bar{x}](\mathcal{M}(x, y) \geq 0.6 \wedge x_1.\text{destination} = y_1.\text{name} \wedge x_2.\text{genre} = y_2.\text{name} \rightarrow (x, \text{likes}, y))$. It enhances the hybrid model $\mathcal{M}$ of [164] by incorporating context features about flights and theaters, and filters FPs with additional logic predicates. That is, although $\mathcal{M}$ suggests to recommend tickets of theater y to user x (*i.e.,* $\mathcal{M}(x, y) \geq 0.6$), if x travels to a city different from y's (during the same season), or if the genre of y does not match the user's preference, then the prediction of $\mathcal{M}$ is FP and is thus overridden.

(b) $\varphi_6 = Q_6[\bar{x}](\mathcal{M}(x, y) < 0.6 \wedge x_3.\text{name} = y_1.\text{name} \wedge x_2.\text{type} = \text{coastal} \wedge y_1.\text{business} = \text{“beach accessories”} \rightarrow (x, \text{likes}, y))$. It reduces FNs of the ML model $\mathcal{M}$ [164] by considering location changes of user x and his social links (specified in Q_4) overlooked by $\mathcal{M}$. It recommends item y to x if x has moved to a coastal city x_2, his friend x_1 likes beach accessories of a particular company x_3, and if y is from the company of x_3, although $\mathcal{M}$ predicts against it.

We defer the full treatment of enriching ML models to a later paper. As will be reported there, this approach improves the accuracy of popular ML models by 20.89% on average, up to 33.10%.

(2) Drug discovery. Drug discovery is a time-consuming and costly process, starting from target selection and validation, through preclinical screening, to clinical trials [79]. On average, the development of a new drug takes 15 years [41] and costs 800 million dollars [8], with a high risk of failure (>90% [17]). To shorten the discovery cycle, reduce the cost and increase the success rate, computational methods have been explored for identifying drug-disease associations (DDA) and drug-drug interaction (DDI).

Fishing Fort has been applied to DDA discovery. Consider dataset CTD (Comparative Toxicogenomics Database) of published curated data about relationships between chemicals (drugs), genes, diseases, and their effect pathways [38]. Modeled as a graph, its vertices represent these entities and edges denote known associations among them. In this context, DDA analysis is equivalent to predicting missing links between drugs and diseases of interest.

Upon the request of partners in pharmacy, Fishing Fort was used to discover GARs targeting Parkinson disease. A simplified GAR mined is $\varphi_7 = Q_7[\bar{x}](X_7 \rightarrow l(x_0, x_1))$, in which $\mathcal{M}_7$ is a pretrained ML model that predicts the associations between genes and disease [125, 167, 188], and pattern Q_7 is depicted in Fig. 5. Together with Q_7, precondition X_7 specifies the following: (1) drug x_0 has a known effect on an inborn genetic blood disease x_2; (2) disease x_1 is a type of Parkinson; (3) drug x_0 interacts with a gene x_3, which shares an effect pathway x_4 with x_1; (4) drug x_0 can interact with a gene x_5, which has an $\mathcal{M}_7$-predicted relationship with x_1 (the dashed arrow in Q_7); and (5) drug x_0 has a known effect on a type of skin cancer x_6, which shares an effect pathway with x_1. The

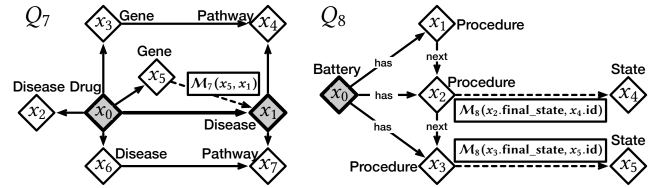

Figure 5: Graph patterns Q_7–Q_8

predicted link $l(x_0, x_1)$ (the bold line in Q_7) indicates that drug x_0 may be associated to Parkinson's disease x_1 in some way.

Such GARs suggest 5 drugs that may have a hidden association with Parkinson's disease. Our partners in pharmacy have verified 4 predictions with published evidence, including Colforsin (Forskolin) [209], Sulindac [34, 162], Tamoxifen [122], and Tretinoin [174]. The remaining one is undergoing their active lab investigation.

(3) Lithium-Ion battery manufacturing. Capacity grading is a critical process in Lithium-Ion battery manufacturing. For safety, only battery cells with roughly the same capacity can be packed in the same module. Thus, cell capacities must be graded with high accuracy. The current industrial practice is conservative: first fully charge every battery, and then measure its capacity with a full discharge. It takes 14+ hours and is energy-consuming. Our industrial partners want to make accurate estimations based on measurements collected during a *partial* charge/discharge.

As a cost-effective solution, Fishing Fort first discretizes the time-series measurements; it splits the entire process into consecutive procedures based on their charging/discharging current. It trains an ML classifier $\mathcal{M}_8$ via unsupervised clustering. Taking as input a vector of battery state statistics (voltage, temperature, and accumulated quantity of electricity), $\mathcal{M}_8$ maps each snapshot of measurements into a set $\mathcal{S}$ of discrete states. The graph G is modeled with three types of vertices: (1) Procedure carries an array of attributes including the procedure ID, its initial/final weights, the initial/final battery state statistics, and the charging/discharging current; (2) State denotes a state in $\mathcal{S}$; and (3) Battery carries metadata of a battery cell, *e.g.,* the cell ID, the testing slot ID, and its capacity interval. An edge denotes either a transition between procedures, or an association between a battery cell and a procedure.

On graphs G, Fishing Fort discovered GARs for capacity grading. A (simplified) GAR is $\varphi_8 = Q_8[\bar{x}](X_8 \rightarrow x_0.\text{capacity} = 8)$, where Q_8 is a pattern shown in Fig. 5, and its consequence grades a matching battery cell as Capacity Interval 8. Together with Q_8, X_8 specifies the following conditions: (1) the weight before and after the Electrolyte Filling procedure (x_1) is 555 ± 25g and 605 ± 25g, respectively; (2) its Formation-A procedure (x_2) uses a constant charging current at 3.8A, with initial voltage between 0–100mV and a final state 324 (x_4) categorized by $\mathcal{M}_8$ (dashed arrows in Q_8); and (3) its Formation-B procedure (x_3) uses a constant charging current at 8.8A, with initial voltage between 3.3–3.4V and final state 738 (x_5).

With such GARs, Fishing Fort reduces charging to 35–50% of the full battery capacity, 75–100% of discharge (in some cases it even avoids discharging completely), and the time for the capacity grading process from 14 hours to 4 hours. With statistics of the partial charge/discharge, it keeps the error rate under 0.4%, a record in the industry. These translate to 80% reduction in energy consumption for charging and cooling, and cut equipment costs in half.

6 OPEN RESEARCH ISSUES

We have demonstrated that each and every of the 4V characterizations of big graphs is a rich source of questions and vitality. As remarked earlier, the study of big graphs has raised as many questions as it has answered. There is naturally much more to be done.

Below we highlight a few topics that demand a full treatment.

(1) Volume. One topic is to study the parallel scalability of computational problems (Section 2.2). What parallel computations can scale with big graphs by adding more machines? For what problems is parallel processing not effective and we have to seek other solutions? Is there a hierarchy of parallel computation complexity classes with reductions and complete problems, when both computational cost and communication cost are taken into account?

Another issue concerns the capacity of a single machine for big graph analytics. Small companies may not afford a 1000-node cluster. To this end several single-machine graph systems have been developed [10, 36, 85, 121, 133, 139, 160, 198, 215]. However, how far can we go with such systems for big graph analytics? Would a problem become parallelly scalable when we adopt multi-core parallelism instead of multi-machine parallelism? For out-of-core systems, how can we systematically optimize CPU-bound and I/O-bound computations? When does a vertex-centric model work better than the graph-centric model, and vice versa? When parallelism alone does not suffice, can we query big graphs under limited resources by making graphs small [62, 63] and queries compact [47]?

(2) Velocity. As shown in Section 3.2, we can deduce an incremental algorithm $\mathcal{A}_\Delta$ from fixpoint batch algorithm $\mathcal{A}$ such that $\mathcal{A}_\Delta$ is correct and bounded relative to $\mathcal{A}$. Can we extend the method and systematically incrementalize graph algorithms beyond fixpoint? Can we develop a practical system to incrementalize algorithms with performance guarantees and with minimum human intervention? Can we incrementalize algorithms beyond graph computations?

Another topic is to give a full treatment of the complexity models of incrementalized algorithms. There has been work on modeling the complexity of incremental computation, in the classical setting [144] and parameterized complexity setting [136]. When it comes to incrementalization, we need to revisit the complexity models in terms of $|\mathsf{AFF}|$, the size of affected areas by updates.

(3) Variety. One topic concerns how to efficiently support semantic joins $S \bowtie_{\mathcal{A}} G$. The evaluation strategy of Section 4.2 requires database systems to invoke HER and attribute/link extraction from graph G. A question is whether we can compute $S \bowtie_{\mathcal{A}} G$ without calling these external functions at runtime?

When S is a relation D in the input database $\mathcal{D}$, we can compute $f(D, G)$ and $h(D, G)$ offline; we reuse them when needed without calling HER and data extraction at runtime, and incrementally maintain them in response to updates to $\mathcal{D}$ and G. As opposed to federated systems, this does not require to store the entire vertex and edge relations of G in a database. However, when S is a subquery Q, it is more challenging. Can we approximate $Q \bowtie_{\mathcal{A}} G$ by pre-computing certain HER match and property relations, and rewriting the semantic join by using the cached relations as an heuristic solution? How accurate is this approximate solution?

Another topic is about incomplete information, a critical issue of data quality [53]. On the one hand, it is common to find attribute values and tuples missing from relational databases. On the other hand, several knowledge graphs are already in place, *e.g.*, FreeBase [23], Yago [98], Wikipedia [5] and DBpedia [123]. These knowledge graphs have accumulated semantic information and expert knowledge about entities. Can we make use of the semantic information of a knowledge graph G to impute missing data in our databases $\mathcal{D}$? This is feasible by leveraging HER to align entities across relations $\mathcal{D}$ and graphs G (Section 4.1). We can also enrich $\mathcal{D}$ with additional attributes and hidden links extracted from G, not limited to filling in null values. Moreover, we can extract data from other source graphs G, not limited to knowledge graphs.

(4) Veracity. As shown in Section 5.1, GARs $Q[\bar{x}](X \to p_0)$ support the primitives of CFDs, DCs and MDs. However, it is intractable to reason about GARs. Worse yet, it is expensive to learn GARs and detect/fix errors in dense graphs when GARs have large patterns Q. Can we find rules that extend relational cleaning rules to graphs, embed ML models as predicates and moreover, allow "tractable" reasoning without degrading the accuracy of ER and CR? In addition, is it possible for such rules to support parametric simulation across diverse graphs, extract data from external sources and impute missing values and missing links? After all, the problem of missing data is more staggering for schemaless graphs than for relational databases. Furthermore, can such rules also deduce temporal orders to cope with stale data? That is, we aim to deal with ER, CR, timeliness and missing data in a uniform framework, while retaining the tractability and accuracy. These also demand revisions to algorithms for rule discovery and error detection/correction.

As shown in Example 4, GARs of the form $Q[\bar{x}](X \to \mathcal{M})$ suggests that we could discover logic conditions X to characterize ML predictions of model $\mathcal{M}$. This is feasible for GNN-based $\mathcal{M}$. It is known that GNN models are at most as expressive as two-variable first-order logic with counting quantifiers $\exists^{\geq p} z$ [25, 90]. Then, can we systematically discover logic interpretation of ML predictions of such models, for vertex classification and link prediction?

A third topic is to explore potential values of big graph analytics. As shown in Section 5.2, Fishing Fort has found encouraging applications in online recommendation, drug discovery and battery manufacturing. It is currently practiced for *target identification* to identify molecular that cures or stops the progression of a disease, *drug repurposing* to treat new diseases with known drugs, and *adverse drug reaction (ADR) prediction* to identify undesirable effects [202]. Extending GARs with temporal graph patterns, it has also proven effective in event prediction when applied to temporal graphs [61], to predict a real-world occurrence that relates to a particular topic and will take place at a specific time [211]. Provided with sufficient data, we expect that it will bring us more surprises from predicting fraud, system failures and disease outbreaks.

ACKNOWLEDGMENTS

The results presented here are from joint work with Yang Cao, Grace Fan, Wenzhi Fu, Ruochun Jin, Muyang Liu, Ping Lu, Chao Tian, Jingbo Xu, Ruiqi Xu, Wenyuan Yu, Qiang Yin, and Jingren Zhou. The author thanks them for their contributions. The author also thanks Shuhao Liu and Chao Tian for their help in preparing examples in this article. The work was supported in part by ERC 652976 and Royal Society Wolfson Research Merit Award WRM/R1/180014.

REFERENCES

- [1] 2020. GraphScope. *https://graphscope.io/*.
- [2] 2021. DBLP collaboration network. *https://snap.stanford.edu/data/com-DBLP.html*.
- [3] 2022. IMDB. *https://www.imdb.com/interfaces*.
- [4] 2022. Neo4J Project. *http://neo4j.org/*.
- [5] 2022. Wikipedia. *https://www.wikipedia.org*.
- [6] Azza Abouzeid, Kamil Bajda-Pawlikowski, Daniel J. Abadi, Alexander Rasin, and Avi Silberschatz. 2009. HadoopDB: An Architectural Hybrid of MapReduce and DBMS Technologies for Analytical Workloads. *PVLDB* 2, 1 (2009), 922–933.
- [7] Umut A. Acar. 2005. *Self-Adjusting Computation*. Ph.D. Dissertation. CMU.
- [8] Christopher P Adams and Van V Brantner. 2006. Estimating the cost of new drug development: is it really 802 million? *Health affairs* 25, 2 (2006), 420–428.
- [9] Sajad Ahmadian, Nima Joorabloo, Mahdi Jalili, Majid Meghdadi, Mohsen Afsharchi, and Yongli Ren. 2018. A temporal clustering approach for social recommender systems. In *ASONAM*. IEEE, 1139–1144.
- [10] Zhiyuan Ai, Mingxing Zhang, Yongwei Wu, Xuehai Qian, Kang Chen, and Weimin Zheng. 2017. Squeezing out All the Value of Loaded Data: An Out-of-core Graph Processing System with Reduced Disk I/O. In *USENIX*. 125–137.
- [11] João Paulo Aires and Felipe Meneguzzi. 2017. Norm Conflict Identification Using Deep Learning. In *AAMAS Workshops*. 194–207.
- [12] Arvind Arasu, Michaela Götz, and Raghav Kaushik. 2010. On active learning of record matching packages. In *SIGMOD*. 783–794.
- [13] Arvind Arasu, Christopher Ré, and Dan Suciu. 2009. Large-Scale Deduplication with Constraints Using Dedupalog. In *ICDE*. 952–963.
- [14] Marcelo Arenas, Leopoldo Bertossi, and Jan Chomicki. 1999. Consistent Query Answers in Inconsistent Databases. In *PODS*. 68–79.
- [15] Zeinab Bahmani, Leopoldo E. Bertossi, and Nikolaos Vasiloglou. 2017. ERBlox: Combining matching dependencies with machine learning for entity resolution. *Int. J. Approx. Reasoning* 83 (2017), 118–141.
- [16] Jørgen Bang-Jensen and Gregory Z. Gutin. 2009. *Digraphs - Theory, Algorithms and Applications, Second Edition*. Springer.
- [17] Nurken Berdigaliyev and Mohamad Aljofan. 2020. An overview of drug discovery and development. *Future Medicinal Chemistry* 12, 10 (2020), 939–947.
- [18] Rianne van den Berg, Thomas N Kipf, and Max Welling. 2017. Graph convolutional matrix completion. *arXiv preprint arXiv:1706.02263* (2017).
- [19] James Bergstra and Yoshua Bengio. 2012. Random search for hyper-parameter optimization. *JMLR* 13, 1 (2012), 281–305.
- [20] Leopoldo E. Bertossi, Solmaz Kolahi, and Laks V. S. Lakshmanan. 2013. Data Cleaning and Query Answering with Matching Dependencies and Matching Functions. *Theory Comput. Syst.* 52, 3 (2013), 441–482.
- [21] Indrajit Bhattacharya and Lise Getoor. 2006. Entity resolution in graphs. *Mining graph data* (2006).
- [22] Dmitry Bogdanov, Martín Haro, Ferdinand Fuhrmann, Anna Xambó, Emilia Gómez, and Perfecto Herrera. 2013. Semantic audio content-based music recommendation and visualization based on user preference examples. *Information Processing & Management* 49, 1 (2013), 13–33.
- [23] Kurt D. Bollacker, Colin Evans, Praveen K. Paritosh, Tim Sturge, and Jamie Taylor. 2008. Freebase: A collaboratively created graph database for structuring human knowledge. In *SIGMOD*. 1247–1250.
- [24] Florian Bourse, Marc Lelarge, and Milan Vojnovic. 2014. Balanced graph edge partition. In *SIGKDD*. 1456–1465.
- [25] Jin-yi Cai, Martin Fürer, and Neil Immerman. 1992. An optimal lower bound on the number of variables for graph identifications. *Comb.* 12, 4 (1992), 389–410.
- [26] Yufei Cai, Paolo G. Giarrusso, Tillmann Rendel, and Klaus Ostermann. 2014. A theory of changes for higher-order languages: Incrementalizing λ-calculi by static differentiation. In *PLDI*. 145–155.
- [27] Zhuhua Cai, Dionysios Logothetis, and Georgos Siganos. 2012. Facilitating real-time graph mining. In *CloudDB*.
- [28] Riccardo Cappuzzo, Paolo Papotti, and Saravanan Thirumuruganathan. 2020. Creating Embeddings of Heterogeneous Relational Datasets for Data Integration Tasks. In *SIGMOD*. 1335–1349.
- [29] Dawei Chen, Cheng Soon Ong, and Lexing Xie. 2016. Learning points and routes to recommend trajectories. In *CIKM*. 2227–2232.
- [30] Xu Chen, Yongfeng Zhang, and Zheng Qin. 2019. Dynamic explainable recommendation based on neural attentive models. In *AAAI*, Vol. 33. 53–60.
- [31] Zhaoqiang Chen, Qun Chen, Fengfeng Fan, Yanyan Wang, Zhuo Wang, Youcef Nafa, Zhanhuai Li, Hailong Liu, and Wei Pan. 2018. Enabling quality control for entity resolution: A human and machine cooperation framework. In *ICDE*. IEEE, 1156–1167.
- [32] Wei-Ta Chu and Ya-Lun Tsai. 2017. A hybrid recommendation system considering visual information for predicting favorite restaurants. *World Wide Web* 20, 6 (2017), 1313–1331.
- [33] Gao Cong, Wenfei Fan, Floris Geerts, Xibei Jia, and Shuai Ma. 2007. Improving Data Quality: Consistency and Accuracy. In *VLDB*. 315–326.
- [34] A Dairam, Edith M Antunes, KS Saravanan, and Santylal Daya. 2006. Non-steroidal anti-inflammatory agents, tolmetin and sulindac, inhibit liver tryptophan 2, 3-dioxygenase activity and alter brain neurotransmitter levels. *Life sciences* 79, 24 (2006), 2269–2274.
- [35] Sanjib Das, Paul Suganthan G. C., AnHai Doan, Jeffrey F. Naughton, Ganesh Krishnan, Rohit Deep, Esteban Arcaute, Vijay Raghavendra, and Youngchoon Park. 2017. Falcon: Scaling Up Hands-Off Crowdsourced Entity Matching to Build Cloud Services. In *SIGMOD*. 1431–1446.
- [36] Roshan Dathathri, Gurbinder Gill, Loc Hoang, Hoang-Vu Dang, Alex Brooks, Nikoli Dryden, Marc Snir, and Keshav Pingali. 2018. Gluon: A Communication-Optimizing Substrate for Distributed Heterogeneous Graph Analytics. In *PLDI*. 752–768.
- [37] Ankur Dave, Alekh Jindal, Li Erran Li, Reynold Xin, Joseph Gonzalez, and Matei Zaharia. 2016. GraphFrames: an integrated API for mixing graph and relational queries. In *GRADES*. 2.
- [38] Allan Peter Davis, Cynthia J Grondin, Robin J Johnson, Daniela Sciaky, Jolene Wiegers, Thomas C Wiegers, and Carolyn J Mattingly. 2021. Comparative toxicogenomics database (CTD): update 2021. *Nucleic acids research* 49, D1 (2021), D1138–D1143.
- [39] Amol Deshpande. 2018. In situ graph querying and analytics with graphgen: Extended abstract. In *GRADES*. 2:1–2:2.
- [40] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *NAACL-HLT*.
- [41] J A. Dimasi. 2001. New drug development in the United States from 1963 to 1999. In *Clinical pharmacology and therapeutics vol. 69,5*.
- [42] Mohamad Dolatshah, Mathew Teoh, Jiannan Wang, and Jian Pei. 2018. Cleaning Crowdsourced Labels Using Oracles For Statistical Classification. *PVLDB* 12, 4 (2018), 376–389.
- [43] Yuxiao Dong, Nitesh V. Chawla, and Ananthram Swami. 2017. metapath2vec: Scalable Representation Learning for Heterogeneous Networks. In *KDD*.
- [44] Jennie Duggan, Aaron J. Elmore, Michael Stonebraker, Magdalena Balazinska, Bill Howe, Jeremy Kepner, Sam Madden, David Maier, Tim Mattson, and Stanley B. Zdonik. 2015. The BigDAWG Polystore System. *SIGMOD Rec.* 44, 2 (2015), 11–16.
- [45] Muhammad Ebraheem, Saravanan Thirumuruganathan, Shafiq R. Joty, Mourad Ouzzani, and Nan Tang. 2018. Distributed Representations of Tuples for Entity Resolution. *PVLDB* 11, 11 (2018), 1454–1467.
- [46] Federico Errica, Marco Podda, Davide Bacciu, and Alessio Micheli. 2020. A Fair Comparison of Graph Neural Networks for Graph Classification. In *ICLR*.
- [47] Grace Fan, Wenfei Fan, Yuanhao Li, Ping Lu, Chao Tian, and Jingren Zhou. 2020. Extending Graph Patterns with Conditions. In *SIGMOD*. 715–729.
- [48] Jing Fan, Adalbert Gerald Soosai Raj, and Jignesh M. Patel. 2015. The Case Against Specialized Graph Analytics Engines. In *CIDR*.
- [49] Wenfei Fan, Zhe Fan, Chao Tian, and Xin Luna Dong. 2015. Keys for graphs. *PVLDB* 8, 12 (2015), 1590–1601.
- [50] Wenfei Fan, Wenzhi Fu, Ruochun Jin, Ping Lu, and Chao Tian. 2022. Discovering Association Rules from Big Graphs. *PVLDB* 15, 7 (2022), 1479–1492.
- [51] Wenfei Fan, Hong Gao, Xibei Jia, Jianzhong Li, and Shuai Ma. 2011. Dynamic constraints for record matching. *VLDB J.* 20, 4 (2011), 495–520.
- [52] Wenfei Fan, Ling Ge, Ruochun Jin, Ping Lu, and Wenyuan Yu. 2022. Linking Entities across Relations and Graphs. In *ICDE*. IEEE.
- [53] Wenfei Fan and Floris Geerts. 2012. *Foundations of Data Quality Management*. Morgan & Claypool Publishers.
- [54] Wenfei Fan, Floris Geerts, Xibei Jia, and Anastasios Kementsietsidis. 2008. Conditional Functional Dependencies for Capturing Data Inconsistencies. *ACM Trans. Database Syst.* 33, 1 (2008), 25:1–25:49.
- [55] Wenfei Fan, Floris Geerts, Nan Tang, and Wenyuan Yu. 2014. Conflict resolution with data currency and consistency. *J. Data and Information Quality* 5, 1-2 (2014), 6:1–6:37.
- [56] Wenfei Fan, Tao He, Longbin Lai, Xue Li, Yong Li, Zhao Li, Zhengping Qian, Chao Tian, Lei Wang, Jingbo Xu, Youyang Yao, Qiang Yin, Wenyuan Yu, Kai Zeng, Kun Zhao, Jingren Zhou, Diwen Zhu, and Rong Zhu. 2021. GraphScope: A Unified Engine For Big Graph Processing. *PVLDB* 14, 12 (2021), 2879–2892.
- [57] Wenfei Fan, Chunming Hu, Xueli Liu, and Ping Lu. 2020. Discovering Graph Functional Dependencies. *ACM Trans. Database Syst.* 45, 3 (2020), 15:1–15:42.
- [58] Wenfei Fan, Chunming Hu, and Chao Tian. 2017. Incremental Graph Computations: Doable and Undoable. In *SIGMOD*. 155–169.
- [59] Wenfei Fan, Ruochun Jin, Muyang Liu, Ping Lu, Xiaojian Luo, Ruiqi Xu, Qiang Yin, Wenyuan Yu, and Jingren Zhou. 2020. Application Driven Graph Partitioning. In *SIGMOD*. 1765–1779.
- [60] Wenfei Fan, Ruochun Jin, Muyang Liu, Ping Lu, Chao Tian, and Jingren Zhou. 2020. Capturing Associations in Graphs. *PVLDB* 13, 11 (2020), 1863–1876.
- [61] Wenfei Fan, Ruochun Jin, Ping Lu, Chao Tian, and Ruiqi Xu. 2022. Towards Event Prediction in Temporal Graphs. *PVLDB* 15, 9 (2022), 1861–1874.
- [62] Wenfei Fan, Yuanhao Li, Muyang Liu, and Can Lu. 2021. Making Graphs Compact by Lossless Contraction. In *SIGMOD*. 472–484.
- [63] Wenfei Fan, Yuanhao Li, Muyang Liu, and Can Lu. 2022. A Hierarchical Contraction Scheme for Querying Big Graphs. In *SIGMOD*. 1726–1740.
- [64] Wenfei Fan, Muyang Liu, Chao Tian, Ruiqi Xu, and Jingren Zhou. 2020. Incrementalization of Graph Partitioning Algorithms. *PVLDB* 13, 8 (2020), 1261–1274.

[65] Wenfei Fan, Xueli Liu, Ping Lu, and Chao Tian. 2018. Catching Numeric Inconsistencies in Graphs. In *SIGMOD*. 381–393.
[66] Wenfei Fan and Ping Lu. 2019. Dependencies for Graphs. *ACM Trans. Database Syst.* 44, 2 (2019), 5:1–5:40.
[67] Wenfei Fan, Ping Lu, Xiaojian Luo, Jingbo Xu, Qiang Yin, Wenyuan Yu, and Ruiqi Xu. 2018. Adaptive Asynchronous Parallelization of Graph Algorithms. In *SIGMOD*. 1141–1156.
[68] Wenfei Fan, Ping Lu, and Chao Tian. 2020. Unifying logic rules and machine learning for entity enhancing. *Sci. China Inf. Sci.* 63, 7 (2020).
[69] Wenfei Fan, Ping Lu, Chao Tian, and Jingren Zhou. 2019. Deducing Certain Fixes to Graphs. *PVLDB* 12, 7 (2019), 752–765.
[70] Wenfei Fan, Ping Lu, Wenyuan Yu, Jingbo Xu, Qiang Yin, Xiaojian Luo, Jingren Zhou, and Ruochun Jin. 2020. Adaptive Asynchronous Parallelization of Graph Algorithms. *ACM Trans. Database Syst.* 45, 2 (2020), 6:1–6:45.
[71] Wenfei Fan and Chao Tian. 2022. Incremental Graph Computations: Doable and Undoable. *ACM Trans. Database Syst.* 47, 2 (2022), 6:1–6:44.
[72] Wenfei Fan, Chao Tian, Yanghao Wang, and Qiang Yin. 2021. Discrepancy Detection and Incremental Detection. *PVLDB* 14, 8 (2021), 1351–1364.
[73] Wenfei Fan, Chao Tian, Ruiqi Xu, Qiang Yin, Wenyuan Yu, and Jingren Zhou. 2021. Incrementalizing Graph Algorithms. In *SIGMOD*. 459–471.
[74] Wenfei Fan, Xin Wang, and Yinghui Wu. 2013. Incremental graph pattern matching. *ACM Trans. Database Syst.* 38, 3 (2013).
[75] Wenfei Fan, Xin Wang, and Yinghui Wu. 2014. Distributed graph simulation: Impossibility and possibility. *PVLDB* 7, 12 (2014), 1083–1094.
[76] Wenfei Fan, Yinghui Wu, and Jingbo Xu. 2016. Functional dependencies for graphs. In *SIGMOD*. 1843–1857.
[77] Wenfei Fan, Jingbo Xu, Yinghui Wu, Wenyuan Yu, Jiaxin Jiang, Zeyu Zheng, Bohan Zhang, Yang Cao, and Chao Tian. 2017. Parallelizing Sequential Graph Computations. In *SIGMOD*. 495–510.
[78] Wenfei Fan, Wenyuan Yu, Jingbo Xu, Jingren Zhou, Xiaojian Luo, Qiang Yin, Ping Lu, Yang Cao, and Ruiqi Xu. 2018. Parallelizing Sequential Graph Computations. *ACM Trans. Database Syst.* 43, 4 (2018), 18:1–18:39.
[79] Chris Fotis, Asier Antoranz, Dimitris Hatziavramidis, Theodore Sakellaropoulos, and Leonidas G. Alexopoulos. 2017. Pathway-based technologies for early drug discovery. *Drug Discovery Today* (2017).
[80] Michael L Fredman and Robert Endre Tarjan. 1987. Fibonacci heaps and their uses in improved network optimization algorithms. *JACM* 34, 3 (1987).
[81] Cheng Fu, Xianpei Han, Le Sun, Bo Chen, Wei Zhang, Suhui Wu, and Hao Kong. 2019. End-to-end multi-perspective matching for entity resolution. In *IJCAI*. 4961–4967.
[82] Xinyu Fu, Jiani Zhang, Ziqiao Meng, and Irwin King. 2020. MAGNN: Metapath aggregated graph neural network for heterogeneous graph embedding. In *WWW*. 2331–2341.
[83] Michael Garey and David Johnson. 1979. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman.
[84] Gartner. 2018. How to create a business case for data quality improvement. *https://www.gartner.com/smarterwithgartner/how-to-create-a-business-case-for-data-quality-improvement/.*
[85] Gurbinder Gill, Roshan Dathathri, Loc Hoang, Ramesh Peri, and Keshav Pingali. 2020. Single Machine Graph Analytics on Massive Datasets Using Intel Optane DC Persistent Memory. *PVLDB* 13, 8 (2020), 1304–1318.
[86] Lukasz Golab, Howard Karloff, Flip Korn, Divesh Srivastava, and Bei Yu. 2008. On generating near-optimal tableaux for conditional functional dependencies. *PVLDB* 1, 1 (2008), 376–390.
[87] Joseph E. Gonzalez, Yucheng Low, Haijie Gu, Danny Bickson, and Carlos Guestrin. 2012. PowerGraph: Distributed Graph-Parallel Computation on Natural Graphs. In *USENIX*. 17–30.
[88] Joseph E. Gonzalez, Reynold S. Xin, Ankur Dave, Daniel Crankshaw, Michael J. Franklin, and Ion Stoica. 2014. GraphX: Graph Processing in a Distributed Dataflow Framework. In *OSDI*. 599–613.
[89] Raymond Greenlaw, H. James Hoover, and Walter L. Ruzzo. 1995. *Limits to Parallel Computation: P-Completeness Theory.* Oxford University Press.
[90] Martin Grohe. 2020. word2vec, node2vec, graph2vec, X2vec: Towards a Theory of Vector Embeddings of Structured Data. In *PODS*. ACM, 1–16.
[91] Songtao Guo, Xin Luna Dong, Divesh Srivastava, and Remi Zajac. 2010. Record Linkage with Uniqueness Constraints and Erroneous Values. *PVLDB* 3, 1 (2010), 417–428.
[92] Daniel Halperin, Victor Teixeira de Almeida, Lee Lee Choo, Shumo Chu, Paraschos Koutris, Dominik Moritz, Jennifer Ortiz, Vaspol Ruamviboonsuk, Jingjing Wang, Andrew Whitaker, Shengliang Xu, Magdalena Balazinska, Bill Howe, and Dan Suciu. 2014. Demonstration of the Myria big data management service. In *SIGMOD*. 881–884.
[93] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. 2017. Neural collaborative filtering. In *WWW*. 173–182.
[94] Xinran He, Junfeng Pan, Ou Jin, Tianbing Xu, Bo Liu, Tao Xu, Yanxin Shi, Antoine Atallah, Ralf Herbrich, Stuart Bowers, and Joaquin Quiñonero Candela. 2014. Practical lessons from predicting clicks on ads at facebook. In *Proceedings of the Eighth International Workshop on Data Mining for Online Advertising*. 1–9.
[95] Alireza Heidari, Joshua McGrath, Ihab F Ilyas, and Theodoros Rekatsinas. 2019. HoloDetect: Few-Shot Learning for Error Detection. In *SIGMOD*. 829–846.
[96] M. R. Henzinger, T. Henzinger, and P. Kopke. 1995. Computing simulations on finite and infinite graphs. In *FOCS*. 453–462.
[97] Linus Hermansson, Tommi Kerola, Fredrik Johansson, Vinay Jethava, and Devdatt Dubhashi. 2013. Entity disambiguation in anonymized graphs using graph kernels. In *CIKM*. 1037–1046.
[98] Johannes Hoffart, Fabian M. Suchanek, Klaus Berberich, and Gerhard Weikum. 2013. YAGO2: A Spatially and Temporally Enhanced Knowledge Base from Wikipedia: Extended Abstract. In *IJCAI*. 3161–3165.
[99] John E. Hopcroft and Richard M. Karp. 1973. An $n^{5/2}$ Algorithm for Maximum Matchings in Bipartite Graphs. *SIAM J. Comput.* 2, 4 (1973), 225–231.
[100] Boyi Hou, Qun Chen, Yanyan Wang, Youcef Nafa, and Zhanhua Li. 2022. Gradual Machine Learning for Entity Resolution. *TKDE* 34, 4 (2022), 1803–1814.
[101] Robert Isele, Anja Jentzsch, and Christian Bizer. 2010. Silk Server - Adding missing Links while consuming Linked Data. In *COLD*, Vol. 665.
[102] Glen Jeh and Jennifer Widom. 2002. Simrank: A measure of structural-context similarity. In *KDD*. 538–543.
[103] Alekh Jindal, Samuel Madden, Malú Castellanos, and Meichun Hsu. 2015. Graph analytics using Vertica relational database. In *BigData*. IEEE Computer Society, 1191–1200.
[104] Alekh Jindal, Praynaa Rawlani, Eugene Wu, Samuel Madden, Amol Deshpande, and Mike Stonebraker. 2014. VERTEXICA: Your Relational Friend for Graph Analytics! *PVLDB* 7, 13 (2014), 1669–1672.
[105] N. D. Jones. 1996. An Introduction to Partial Evaluation. *Comput. Surveys* 28, 3 (1996), 480–503.
[106] Richard M. Karp and Vijaya Ramachandran. 1988. *A Survey of Parallel Algorithms for Shared-Memory Machines.* Technical Report UCB/CSD-88-408. EECS Department, University of California, Berkeley. http://www.eecs.berkeley.edu/Pubs/TechRpts/1988/5865.html
[107] George Karypis and Vipin Kumar. 1998. Multilevel k-way Partitioning Scheme for Irregular Graphs. *J. Parallel Distributed Comput.* 48, 1 (1998), 96–129.
[108] Jungo Kasai, Kun Qian, Sairam Gurajada, Yunyao Li, and Lucian Popa. 2019. Low-resource deep entity resolution with transfer and active learning. *arXiv preprint arXiv:1906.08042* (2019).
[109] Jeremy Kepner, William Arcand, William Bergeron, Nadya T. Bliss, Robert Bond, Chansup Byun, Gary Condon, Kenneth Gregson, Matthew Hubbell, Jonathan Kurz, Andrew McCabe, Peter Michaleas, Andrew Prout, Albert Reuther, Antonio Rosa, and Charles Yee. 2012. Dynamic distributed dimensional data model (D4M) database and computation system. In *ICASSP*. IEEE, 5349–5352.
[110] Zuhair Khayyat, Ihab F. Ilyas, Alekh Jindal, Samuel Madden, Mourad Ouzzani, Paolo Papotti, Jorge-Arnulfo Quiané-Ruiz, Nan Tang, and Si Yin. 2015. BigDansing: A System for Big Data Cleansing. In *SIGMOD*. 1215–1230.
[111] Mijung Kim and K Selçuk Candan. 2012. SBV-Cut: Vertex-cut based graph partitioning using structural balance vertices. *DKE* 72 (2012), 285–303.
[112] Boyan Kolev, Carlyna Bondiombouy, Patrick Valduriez, Ricardo Jiménez-Peris, Raquel Pau, and José Pereira. 2016. The CloudMdsQL, Multistore System. In *SIGMOD*. ACM, 2113–2116.
[113] Pradap Konda, Sanjib Das, Paul Suganthan G. C., AnHai Doan, Adel Ardalan, Jeffrey R. Ballard, Han Li, Fatemah Panahi, Haojun Zhang, Jeffrey F. Naughton, Shishir Prasad, Ganesh Krishnan, Rohit Deep, and Vijay Raghavendra. 2016. Magellan: Toward building entity matching management systems. *PVLDB* 9, 12 (2016), 1197–1208.
[114] Hanna Köpcke, Andreas Thor, and Erhard Rahm. 2010. Evaluation of entity resolution approaches on real-world match problems. *PVLDB* 3, 1 (2010), 484–493.
[115] Yehuda Koren. 2008. Factorization meets the neighborhood: a multifaceted collaborative filtering model. In *SIGKDD*. 426–434.
[116] Yehuda Koren, Robert Bell, and Chris Volinsky. 2009. Matrix factorization techniques for recommender systems. *Computer* 42, 8 (2009), 30–37.
[117] Christos Koutras, Marios Fragkoulis, Asterios Katsifodimos, and Christoph Lofi. 2020. REMA: Graph Embeddings-based Relational Schema Matching. In *EDBT/ICDT*, Vol. 2578.
[118] Clyde P. Kruskal, Larry Rudolph, and Marc Snir. 1990. A Complexity Theory of Efficient Parallel Algorithms. *Theor. Comput. Sci.* 71, 1 (1990), 95–132.
[119] Mitsuru Kusumoto, Takanori Maehara, and Ken-ichi Kawarabayashi. 2014. Scalable similarity search for SimRank. In *SIGMOD*. 325–336.
[120] Selasi Kwashie, Lin Liu, Jixue Liu, Markus Stumptner, Jiuyong Li, and Lujing Yang. 2019. Certus: An effective entity resolution approach with graph differential dependencies (GDDs). *PVLDB* 12, 6 (2019), 653–666.
[121] Aapo Kyrola, Guy E. Blelloch, and Carlos Guestrin. 2012. GraphChi: Large-scale graph computation on just a PC. In *OSDI*. 31–46.
[122] Jeanne C Latourelle, Merete Dybdahl, Anita L Destefano, Richard H Myers, and Timothy L Lash. 2010. Risk of Parkinson's disease after tamoxifen treatment. *BMC neurology* 10, 1 (2010), 1–7.
[123] Jens Lehmann, Robert Isele, Max Jakob, Anja Jentzsch, Dimitris Kontokostas, Pablo N. Mendes, Sebastian Hellmann, Mohamed Morsey, Patrick van Kleef, Sören Auer, and Christian Bizer. 2015. DBpedia - A large-scale, multilingual

knowledge base extracted from Wikipedia. *Semantic Web* 6, 2 (2015), 167–195.
[124] Bing Li, Wei Wang, Yifang Sun, Linhan Zhang, Muhammad Asif Ali, and Yi Wang. 2020. GraphER: Token-Centric Entity Resolution with Graph Convolutional Neural Networks.. In *AAAI*. 8172–8179.
[125] Yu Li, Hiroyuki Kuwahara, Peng Yang, Le Song, and Xin Gao. 2019. PGCN: Disease gene prioritization by disease and gene embedding through graph convolutional neural networks. *biorxiv* (2019), 532226.
[126] Yuliang Li, Jinfeng Li, Yoshihiko Suhara, AnHai Doan, and Wang-Chiew Tan. 2020. Deep Entity Matching with Pre-Trained Language Models. *PVLDB* 14, 1 (2020), 50–60.
[127] Leonid Libkin. 2004. *Elements of Finite Model Theory*. Springer.
[128] Greg Linden, Brent Smith, and Jeremy York. 2003. Amazon. com recommendations: Item-to-item collaborative filtering. *IEEE Internet computing* 7, 1 (2003), 76–80.
[129] Yanhong A. Liu. 2000. Efficiency by Incrementalization: An Introduction. *High. Order Symb. Comput.* 13, 4 (2000), 289–313.
[130] Beth Logan and Ariel Salomon. 2001. A content-based music similarity function. *Cambridge Research Labs-Tech Report* (2001).
[131] Yucheng Low, Joseph Gonzalez, Aapo Kyrola, Danny Bickson, Carlos Guestrin, and Joseph M. Hellerstein. 2012. Distributed GraphLab: A Framework for Machine Learning in the Cloud. *PVLDB* 5, 8 (2012), 716–727.
[132] Xin Luo, Mengchu Zhou, Yunni Xia, and Qingsheng Zhu. 2014. An efficient non-negative matrix-factorization-based approach to collaborative filtering for recommender systems. *IEEE Transactions on Industrial Informatics* 10, 2 (2014), 1273–1284.
[133] Steffen Maass, Changwoo Min, Sanidhya Kashyap, Woon-Hak Kang, Mohan Kumar, and Taesoo Kim. 2017. Mosaic: Processing a Trillion-Edge Graph on a Single Machine. In *EuroSys*. ACM, 527–543.
[134] Mohammad Mahdavi, Ziawasch Abedjan, Raul Castro Fernandez, Samuel Madden, Mourad Ouzzani, Michael Stonebraker, and Nan Tang. 2019. Raha: A Configuration-Free Error Detection System. In *SIGMOD*. 865–882.
[135] Grzegorz Malewicz, Matthew H. Austern, Aart J. C. Bik, James C. Dehnert, Ilan Horn, Naty Leiser, and Grzegorz Czajkowski. 2010. Pregel: A system for large-scale graph processing. In *SIGMOD*. 135–146.
[136] Bernard Mans and Luke Mathieson. 2017. Incremental Problems in the Parameterized Complexity Setting. *Theory Comput. Syst.* 60, 1 (2017), 3–19.
[137] Mugilan Mariappan and Keval Vora. 2019. GraphBolt: Dependency-Driven Synchronous Processing of Streaming Graphs. In *EuroSys*. 25:1–25:16.
[138] Brian McFee, Luke Barrington, and Gert Lanckriet. 2012. Learning content similarity for music recommendation. *IEEE transactions on audio, speech, and language processing* 20, 8 (2012), 2207–2218.
[139] Frank McSherry, Michael Isard, and Derek Gordon Murray. 2015. Scalability! But at what COST?. In *HotOS*.
[140] Frank McSherry, Derek Gordon Murray, Rebecca Isaacs, and Michael Isard. 2013. Differential Dataflow. In *CIDR*.
[141] Alberto O. Mendelzon and Peter T. Wood. 1995. Finding Regular Simple Paths in Graph Databases. *SIAM J. Comput.* 24, 6 (1995), 1235–1258.
[142] Franck Michel, Johan Montagnat, and Catherine Faron Zucker. 2014. A survey of RDB to RDF translation approaches and tools. *https://hal.archives-ouvertes.fr/hal-00903568/file/Rapport_Rech_I3S_v2_-_Michel_et_al_2013_-_A_survey_of_RDB_to_RDF_translation_approaches_and_tools.pdf*.
[143] Robin Milner. 1989. *Communication and Concurrency*. Prentice Hall.
[144] Peter Bro Miltersen, Sairam Subramanian, Jeffrey Scott Vitter, and Roberto Tamassia. 1994. Complexity Models for Incremental Computation. *Theor. Comput. Sci.* 130, 1 (1994), 203–236.
[145] Sidharth Mudgal, Han Li, Theodoros Rekatsinas, AnHai Doan, Youngchoon Park, Ganesh Krishnan, Rohit Deep, Esteban Arcaute, and Vijay Raghavendra. 2018. Deep Learning for Entity Matching: A Design Space Exploration. In *SIGMOD*. 19–34.
[146] Fatemeh Nargesian, Erkang Zhu, Renée J. Miller, Ken Q. Pu, and Patricia C. Arocena. 2019. Data Lake Management: Challenges and Opportunities. *PVLDB* 12, 12 (2019), 1986–1989.
[147] Axel-Cyrille Ngonga Ngomo and Sören Auer. 2011. LIMES - A Time-Efficient Approach for Large-Scale Link Discovery on the Web of Data. In *IJCAI*. 2312–2317.
[148] Donald Nguyen, Andrew Lenharth, and Keshav Pingali. 2013. A lightweight infrastructure for graph analytics. In *SOSP*. 456–471.
[149] Phuc Nguyen, Ikuya Yamada, Natthawut Kertkeidkachorn, Ryutaro Ichise, and Hideaki Takeda. 2020. MTab4Wikidata at SemTab 2020: Tabular Data Annotation with Wikidata. In *SemTabISWC*, Vol. 2775. 86–95.
[150] George Papadakis, Leonidas Tsekouras, Emmanouil Thanos, George Giannakopoulos, Themis Palpanas, and Manolis Koubarakis. 2018. The return of JedAI: End-to-end entity resolution for structured and semi-structured data. *PVLDB* 11, 12 (2018), 1950–1953.
[151] Christos H. Papadimitriou. 1994. *Computational complexity*. Addison-Wesley.
[152] Kun Qian, Lucian Popa, and Prithviraj Sen. 2017. Active Learning for Large-Scale Entity Resolution. In *CIKM*. 1379–1388.
[153] Abdul Quamar and Amol Deshpande. 2016. NScaleSpark: subgraph-centric graph analytics on Apache Spark. In *NDA*. ACM, 5:1–5:8.
[154] G. Ramalingam and Thomas Reps. 1996. An incremental algorithm for a generalization of the shortest-path problem. *J. Algorithms* 21, 2 (1996), 267–305.
[155] G. Ramalingam and Thomas Reps. 1996. On the computational complexity of dynamic graph problems. *ACM Trans. Database Syst.* 158, 1-2 (1996), 233–277.
[156] G. Ramalingam and Thomas W. Reps. 1996. An Incremental Algorithm for a Generalization of the Shortest-Path Problem. *J. Algorithms* 21, 2 (1996), 267–305.
[157] Thomas C. Redman. 2016. Bad Data Costs the U.S. $3 Trillion Per Year. Harvard Business Review. *https://hbr.org/2016/09/bad-data-costs-the-u-s-3-trillion-per-year*.
[158] Nils Reimers and Iryna Gurevych. 2019. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. In *EMNLP-IJCNLP*. 3980–3990.
[159] Theodoros Rekatsinas, Xu Chu, Ihab F. Ilyas, and Christopher Ré. 2017. HoloClean: Holistic Data Repairs with Probabilistic Inference. *PVLDB* 10, 11 (2017), 1190–1201.
[160] Amitabha Roy, Ivo Mihailovic, and Willy Zwaenepoel. 2013. X-stream: Edge-centric graph processing using streaming partitions. In *SOSP*. ACM, 472–488.
[161] Alieh Saeedi, Eric Peukert, and Erhard Rahm. 2018. Using link features for entity clustering in knowledge graphs. In *ESWC*. 576–592.
[162] R Sandyk and MA Gillman. 1985. Acute exacerbation of Parkinson's disease with sulindac. *Annals of neurology* 17, 1 (1985), 104–105.
[163] Jan Schluter and Christian Osendorfer. 2011. Music similarity estimation with the mean-covariance restricted Boltzmann machine. In *ICMLA*, Vol. 2. IEEE, 118–123.
[164] Shilad Sen, Jesse Vig, and John Riedl. 2009. Tagommenders: connecting users to items through tags. In *WWW*. 671–680.
[165] Bin Shao, Haixun Wang, and Yatao Li. 2013. Trinity: a distributed graph engine on a memory cloud. In *SIGMOD*. 505–516.
[166] Shenzhen Institute of Computing Sciences. 2022. Fishing Fort. *https://en.sics.ac.cn/col84/index*.
[167] Juan Shu, Yu Li, Sheng Wang, Bowei Xi, and Jianzhu Ma. 2021. Disease gene prediction with privileged information and heteroscedastic dropout. *Bioinformatics* 37, Supplement_1 (2021), i410–i417.
[168] Benjamin A. Steer, Alhamza Alnaimi, Marco A. B. F. G. Lotz, Félix Cuadrado, Luis M. Vaquero, and Joan Varvenne. 2017. Cytosm: Declarative Property Graph Queries Without Data Migration. In *GRADES*. 4:1–4:6.
[169] Fabian M. Suchanek, Serge Abiteboul, and Pierre Senellart. 2011. PARIS: Probabilistic Alignment of Relations, Instances, and Schema. *PVLDB* 5, 3 (2011), 157–168.
[170] Huifeng Sun, Yong Peng, Junliang Chen, Chuanchang Liu, and Yuzhuo Sun. 2011. A New Similarity Measure Based on Adjusted Euclidean Distance for Memory-based Collaborative Filtering. *JSW* 6, 6 (2011), 993–1000.
[171] Yizhou Sun, Jiawei Han, Xifeng Yan, Philip S. Yu, and Tianyi Wu. 2011. PathSim: Meta Path-Based Top-K Similarity Search in Heterogeneous Information Networks. *PVLDB* 4, 11 (2011), 992–1003.
[172] Zhu Sun, Qing Guo, Jie Yang, Hui Fang, Guibing Guo, Jie Zhang, and Robin Burke. 2019. Research commentary on recommendations with side information: A survey and research directions. *Electronic Commerce Research and Applications* 37 (2019).
[173] Katia P. Sycara. 1993. Machine learning for intelligent support of conflict resolution. *Decision Support Systems* 10, 2 (1993), 121–136.
[174] Bo-Tao Tan, Li Wang, Sen Li, Zai-Yun Long, Ya-Min Wu, and Yuan Liu. 2015. Retinoic acid induced the differentiation of neural stem cells from embryonic spinal cord into functional neurons in vitro. *International journal of clinical and experimental pathology* 8, 7 (2015).
[175] Nan Tang, Ju Fan, Fangyi Li, Jianhong Tu, Xiaoyong Du, Guoliang Li, Samuel Madden, and Mourad Ouzzani. 2021. RPT: Relational Pre-trained Transformer Is Almost All You Need towards Democratizing Data Preparation. *PVLDB* 14, 8 (2021), 1254–1261.
[176] Robert Endre Tarjan. 1972. Depth-First Search and Linear Graph Algorithms. *SIAM J. Comput.* 1, 2 (1972), 146–160.
[177] Tim Teitelbaum and Thomas W. Reps. 1981. The Cornell Program Synthesizer: A Syntax-Directed Programming Environment. *Commun. ACM* 24, 9 (1981), 563–573.
[178] Yuanyuan Tian, Andrey Balmin, Severin Andreas Corsten, and John McPherson Shirish Tatikonda. 2013. From "Think Like a Vertex" to "Think Like a Graph". *PVLDB* 7, 7 (2013), 193–204.
[179] Rakshit Trivedi, Bunyamin Sisman, Jun Ma, Christos Faloutsos, Hongyuan Zha, and Xin Luna Dong. 2018. Linknbed: Multi-graph representation learning with entity linkage. In *ACL*.
[180] Shalini Tyagi and Ernesto Jimenez-Ruiz. 2020. LexMa: Tabular data to knowledge graph matching using lexical techniques. In *CEUR Workshop Proceedings*, Vol. 2775. 59–64.
[181] Farman Ullah, Ghulam Sarwar, Sung Chang Lee, Yun Kyung Park, Kyeong Deok Moon, and Jin Tae Kim. 2012. Hybrid recommender system with temporal information. In *ICOIN*. IEEE, 421–425.
[182] Leslie G. Valiant. 1990. A Bridging Model for Parallel Computation. *Commun.*

ACM 33, 8 (1990), 103–111.
[183] Aäron Van Den Oord, Sander Dieleman, and Benjamin Schrauwen. 2013. Deep content-based music recommendation. In *NIPS*. Neural Information Processing Systems Foundation (NIPS), 2643–2651.
[184] Larysa Visengeriyeva and Ziawasch Abedjan. 2018. Metadata-driven error detection. In *SSDBM*. 1:1–1:12.
[185] Keval Vora, Rajiv Gupta, and Guoqing (Harry) Xu. 2017. KickStarter: Fast and Accurate Computations on Streaming Graphs via Trimmed Approximations. In *ASPLOS*.
[186] W3C. 2012. Relational Databases to RDF (RDB2RDF).
[187] Guozhang Wang, Wenlei Xie, Alan J. Demers, and Johannes Gehrke. 2013. Asynchronous Large-Scale Graph Processing Made Easy. In *CIDR*.
[188] Xiaochan Wang, Yuchong Gong, Jing Yi, and Wen Zhang. 2019. Predicting gene-disease associations from the heterogeneous network using graph embedding. In *IEEE International conference on bioinformatics and biomedicine (BIBM)*. 504–511.
[189] Xiang Wang, Xiangnan He, Meng Wang, Fuli Feng, and Tat-Seng Chua. 2019. Neural graph collaborative filtering. In *SIGIR*. 165–174.
[190] Xinxi Wang and Ye Wang. 2014. Improving content-based and hybrid music recommendation using deep learning. In *ACM Multimedia*. 627–636.
[191] Yue Wang, Zhe Wang, Ziyuan Zhao, Zijian Li, Xun Jian, Hao Xin, Lei Chen, Jianchun Song, Zhenhong Chen, and Meng Zhao. 2022. Effective Similarity Search on Heterogeneous Networks: A Meta-path Free Approach. *TKDE* 34, 7 (2022), 3225–3240.
[192] Duncan J Watts and Steven H Strogatz. 1998. Collective dynamics of 'small-world' networks. *Nature* 393, 6684 (1998), 440–442.
[193] Steven Euijong Whang and Hector Garcia-Molina. 2013. Joint entity resolution on multiple datasets. *The VLDB Journal* 22, 6 (2013), 773–795.
[194] Charith Wickramaarachchi, Charalampos Chelmis, and Viktor K. Prasanna. 2015. Empowering Fast Incremental Computation over Large Scale Dynamic Graphs. In *IPDPS*.
[195] Renzhi Wu, Sanya Chaba, Saurabh Sawlani, Xu Chu, and Saravanan Thirumuruganathan. 2020. ZeroER: Entity Resolution using Zero Labeled Examples. In *SIGMOD*. 1149–1164.
[196] Yuting Wu, Xiao Liu, Yansong Feng, Zheng Wang, Rui Yan, and Dongyan Zhao. 2019. Relation-Aware Entity Alignment for Heterogeneous Knowledge Graphs. In *IJCAI*. 5278–5284.
[197] Chenning Xie, Rong Chen, Haibing Guan, Binyu Zang, and Haibo Chen. 2015. SYNC or ASYNC: Time to fuse for distributed graph-parallel computation. In *PPOPP*. 194–204.
[198] Xianghao Xu, Fang Wang, Hong Jiang, Yongli Cheng, Dan Feng, and Yongxuan Zhang. 2020. A Hybrid Update Strategy for I/O-Efficient Out-of-Core Graph Processing. *IEEE Trans. Parallel Distributed Syst.* 31, 8 (2020), 1767–1782.
[199] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L Hamilton, and Jure Leskovec. 2018. Graph convolutional neural networks for web-scale recommender systems. In *SIGKDD*. 974–983.
[200] Weiren Yu, Xuemin Lin, Wenjie Zhang, Jian Pei, and Julie A McCann. 2019. SimRank*: effective and scalable pairwise similarity search based on graph topology. *The VLDB Journal* 28, 3 (2019), 401–426.
[201] Timothy A. K. Zakian, Ludovic A. R. Capelli, and Zhenjiang Hu. 2019. Incrementalization of Vertex-Centric Programs. In *IPDPS*.
[202] Xiangxiang Zeng, Xinqi Tu, Yuansheng Liu, Xiangzheng Fu, and Yansen Su. 2022. Toward better drug discovery with knowledge graph. *Current opinion in structural biology* 72 (2022), 114–126.
[203] Baichuan Zhang and Mohammad Al Hasan. 2017. Name disambiguation in anonymized graphs using network embedding. In *CIKM*. 1239–1248.
[204] Bingjun Zhang, Jialie Shen, Qiaoliang Xiang, and Ye Wang. 2009. Compositemap: a novel framework for music similarity measure. In *SIGIR*. 403–410.
[205] Dongxiang Zhang, Long Guo, Xiangnan He, Jie Shao, Sai Wu, and Heng Tao Shen. 2018. A Graph-Theoretic Fusion Framework for Unsupervised Entity Resolution. In *ICDE*. 713–724.
[206] Qingheng Zhang, Zequn Sun, Wei Hu, Muhao Chen, Lingbing Guo, and Yuzhong Qu. 2019. Multi-view Knowledge Graph Embedding for Entity Alignment. In *IJCAI*. 5429–5435.
[207] Shuo Zhang, Edgar Meij, Krisztian Balog, and Ridho Reinanda. 2020. Novel Entity Discovery from Web Tables. In *WWW*. 1298–1308.
[208] Chen Zhao and Yeye He. 2019. Auto-EM: End-to-end Fuzzy Entity-Matching using Pre-trained Deep Models and Transfer Learning. In *WWW*. 2413–2424.
[209] Jie Zhao, Manish Kumar, Jeevan Sharma, and Zhihai Yuan. 2021. Arbutin effectively ameliorates the symptoms of Parkinson's disease: The role of adenosine receptors and cyclic adenosine monophosphate. *Neural regeneration research* 16, 10 (2021), 2030.
[210] Kangfei Zhao and Jeffrey Xu Yu. 2017. All-in-One: Graph Processing in RDBMSs Revisited. In *SIGMOD*. 1165–1180.
[211] Liang Zhao. 2021. Event Prediction in the Big Data Era: A Systematic Survey. *ACM Comput. Surv.* 54, 5 (2021), 94:1–94:37.
[212] Zhongying Zhao, Xuejian Zhang, Hui Zhou, Chao Li, Maoguo Gong, and Yongqing Wang. 2020. HetNERec: Heterogeneous network embedding based recommendation. *Knowledge-Based Systems* 204 (2020).
[213] Lei Zheng, Vahid Noroozi, and Philip S Yu. 2017. Joint deep modeling of users and items using reviews for recommendation. In *WSDM*. 425–434.
[214] Minpeng Zhu and Tore Risch. 2011. Querying combined cloud-based and relational databases. In *CSC*. 330–335.
[215] Xiaowei Zhu, Wentao Han, and Wenguang Chen. 2015. GridGraph: Large-Scale Graph Processing on a Single Machine Using 2-Level Hierarchical Partitioning. In *USENIX ATC*. 375–386.
[216] Cai-Nicolas Ziegler, Georg Lausen, and Lars Schmidt-Thieme. 2004. Taxonomy-driven computation of product recommendations. In *CIKM*. 406–415.