MiniClean: A Single-Machine System for Cleaning Big Graphs

Wenchao Bai
Southeast University
Nanjing, China
wbai@seu.edu.cn

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

Jiahui Jin
Southeast University
Nanjing, China
jjin@seu.edu.cn

Daji Li
Jian Li
Shenzhen Institute of Computing Sciences
Shenzhen, China

Shuhao Liu
Shenzhen Institute of Computing Sciences
Shenzhen, China
shuhao@sics.ac.cn

Mingliang Ouyang
Qiang Yuan
Shenzhen Institute of Computing Sciences
Shenzhen, China

Abstract

We demonstrate MiniClean, a single-machine system for cleaning large-scale graphs. MiniClean adopts a rule-based approach that unifies logic reasoning and machine learning, and supports rule discovery, error detection and error correction. To cope with repeated graph pattern matching and large intermediate results, MiniClean proposes (1) a pipelined workflow that coordinates CPU, GPU and I/O operations, (2) memory footprint reduction by bundled processing and data compression, and (3) a multi-mode parallel model that combines SIMD, pipelined, and independent parallelism to maximize CPU–GPU synergy. We demonstrate how MiniClean cleans real-life billion-scale graphs and outperforms a SOTA multi-machine system with a 32-node cluster by at least 8.09×.

CCS Concepts

• **Information systems → Data cleaning**; **Graph-based database models**.

Keywords

Graph Cleaning Rules; Single Machine Systems; GPU

ACM Reference Format:
Wenchao Bai, Wenfei Fan, Jiahui Jin, Daji Li, Jian Li, Shuhao Liu, Mingliang Ouyang, and Qiang Yuan. 2025. MiniClean: A Single-Machine System for Cleaning Big Graphs. In *Companion of the 2025 International Conference on Management of Data (SIGMOD-Companion '25), June 22–27, 2025, Berlin, Germany*. ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3722212.3725115

1 Introduction

Graphs have found prevalent use in drug discovery, battery manufacturing, online recommendation, and fraud detection, among other things [10]. However, real-life graphs often bear errors, having duplicates (*e.g.,* nodes representing the same entity) and conflicts (*e.g.,* contradictory facts). In light of this, graph cleaning is a must for, *e.g.,* drug discovery [6], to detect and correct errors.

A recent approach to cleaning graphs is based on Graph Cleaning Rules (GCRs) [5]. GCRs may embed machine learning (ML) models as predicates, and unify logic reasoning and ML predictions to improve the accuracy of error detection and correction. Better still, GCRs can explain errors detected and corrections to errors in logic, while "reliable explanations are critical" to drug discovery [6].

No matter how desirable, graph cleaning with GCRs is costly. With a 32-node cluster, a state-of-the-art (SOTA) multi-machine graph system takes 2.5h to discover GCRs and 1.9h to correct errors in a graph of 4M nodes and 26.5M edges. Moreover, there is increasing need for single-machine graph-cleaning systems in *e.g.,* industrial Internet-of-Things and edge computing, where deploying multi-machine clusters is often beyond reach in practice. However, while SOTA single-machine graph systems work well on light-weight analytics such as graph connectivity and PageRank, none can discover rules or correct errors even in small graphs.

Graph cleaning with GCRs poses several unique challenges.

(1) Computation-heavy. Cleaning a graph G with GCRs involves repeated enumeration of matches of graph patterns in G (see Section 2), a task far more computationally intensive than graph connectivity and PageRank. Even with GPUs, cleaning a million-scale graph requires processing times in the order of hours. Moreover, the use of GPU introduces challenges to pipelining and balancing I/O, CPU/GPU operations, and the cost of CPU–GPU data transfers. Finding an optimal schedule for I/O, CPU and GPU is NP-complete.

(2) Excessive intermediate results. Rule discovery, error detection and error correction may generate a large volume of intermediate results, such as matches of various patterns. Simply swapping data between memory and secondary storage leads to excessive I/O,

and can cause crashes in systems requiring results to fit entirely in memory [11]. We will show that single-machine systems typically exhaust memory or exceed time bounds on moderately sized graphs, as I/O remains the primary bottleneck due to repetitive loading and eviction of intermediate data in memory. A more sophisticated strategy for memory management is required.

(3) Parallel model. Existing graph systems typically adopt a fixed parallel model, *e.g., vertex-centric* (VC) [7], *edge-centric* (EC) [11], or a hybrid [8] of VC and graph-centric (GC) [4]. However, as will be demonstrated, a single parallel model no longer suffices to effectively manage CPU–GPU collaboration for graph cleaning tasks. CPUs, with limited parallelism, can handle flexible control flows with diverse data dependencies in irregular graph structures. In contrast, GPUs, with their massive SIMD (Single Machine Multiple Data) parallelism, require parallel algorithms designed for regular data structures and control flows. These call for an adaptive parallel model to maximize the resource utilization of a single machine.

MiniClean (Section 2). In response to the practical need, we have developed MiniClean, a GPU-accelerated single-machine system for cost-effective graph cleaning. It features the following.

(1) Parallel graph cleaning. MiniClean proposes a two-stage workflow for enumerating matches of graph patterns, handling irregular graph structures and massive parallel processing on CPU and GPU, respectively. In this way, it pipelines I/O, CPU, and GPU operations for effective workload distribution and higher resource utilization.

(2) Host-side optimizations. To reduce repetitive computations, intermediate data sizes, and CPU–GPU data transfers, MiniClean adopts a combination of strategies, including (a) a bundling strategy that groups similar pattern matching tasks, (b) an intermediate data compression technique using conditional succinct tables, and (c) an adaptive bundling strategy to balance CPU and GPU workloads.

(3) A multi-mode parallel model. MiniClean integrates (a) pipelined parallelism by overlapping CPU and GPU operations; (b) SIMD parallelism by leveraging the concurrent instruction execution capability of GPUs; and (c) independent parallelism by executing multiple cleaning tasks simultaneously. To address the intractable scheduling problem and optimize resource utilization, it proposes an effective heuristic scheduler to minimize I/O and data transfers.

Demonstration (Section 3). Using real-life graphs, we will give a guided tour of MiniClean, from rule discovery, error detection to error correction. Participants are invited to interact with MiniClean to evaluate the effectiveness of individual optimization techniques and their integration. They will witness how MiniClean cleans billion-scale graphs with a single machine, significantly outperforming SOTA multi-machine systems with 32 nodes. In contrast, the SOTA single-machine systems are unable to clean even small graphs.

2 MiniClean: System Overview

This section reviews GCR-based graph cleaning (Section 2.1) and details MiniClean's design (Section 2.2–2.4).

2.1 Parallel Graph Cleaning with GCRs

Preliminaries. A graph is $G = (V, E, L, F_A)$, where (a) V is a finite set of nodes; (b) E is a finite set of edges in which $e = (v, l, v')$ is an edge labeled l from node v to v'; (c) each node $v \in V$ has label $L(v)$, and carries a finite set $F_A(v) = (A_1, \ldots, A_n)$ of *attributes*.

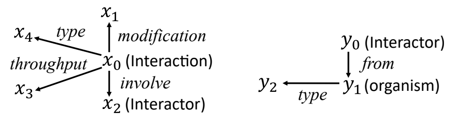

Figure 1: A dual pattern of sample GCRs, discovered from BioGRID.

A *star pattern* is $Q[x_0, \bar{x}] = (V_Q, E_Q, L_Q, \mu)$, where (1) V_Q (resp. E_Q) is a set of pattern nodes (resp. edges); (2) L_Q assigns a label to each vertex in V_Q; (3) $\bar{x}$ is a list of distinct variables, and μ is a bijective mapping from $\bar{x}$ to the nodes of Q; (4) x_0 is a designated variable in $\bar{x}$, referred to as the *center* of Q; and (5) for each $z \in \bar{x}$, there exists a single path from x_0 to z, z has at most one child, except x_0. That is, $Q[x_0, \bar{x}]$ forms a star shape with x_0 as the center and paths linking x_0 to *leaves* (*i.e.,* nodes without any child).

A *dual pattern* $Q[x_0, y_0] = \langle Q_x[x_0, \bar{x}], Q_y[y_0, \bar{y}]\rangle$ consists of two disjoint star patterns $Q_x[x_0, \bar{x}]$ and $Q_y[y_0, \bar{y}]$ that share no common nodes. Intuitively, Q represents two entities x_0 and y_0 with (possibly) heterogeneous structures in a schemaless graph. The star patterns specify features for pairwise comparison between x_0 and y_0.

GCRs. Over dual pattern $Q[x_0, y_0]$, a *predicate* is defined as

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

where $\oplus$ is one of $=, \neq, <, \leq, >, \geq$; $x \in \bar{x}$ and $y \in \bar{y}$ are variables in star patterns Q_x and Q_y, respectively, and $z \in \bar{x} \cup \bar{y}$; c is a constant; A and B are attributes; $x.\bar{A}$ is a list of attributes at x; and $\mathcal{M}(\cdot,\cdot)$ is an ML model that returns true iff $\mathcal{M}$ predicts true at $(x.\bar{A}, y.\bar{B})$ for, *e.g.,* similarity checking, entity resolution, and link prediction.

A *graph cleaning rule* (GCR) φ has the following form:

$$Q[x_0, y_0](X \rightarrow p_0),$$

where $Q[x_0, y_0]$ is a dual pattern, X is a conjunction of predicates of Q, and p_0 is a predicate of Q. In X, (1) *binary predicates* $x.A \oplus y.B$ and $\mathcal{M}(\cdot,\cdot)$ are defined on either two leaves or the centers x and y in the two stars of Q, respectively; and (2) each leaf may carry at most binary predicate but multiple *unary predicates* $z.A \oplus c$.

Intuitively, Q is a dual pattern that identifies relevant entities, and $X \rightarrow p_0$ is a dependency that discloses the correlations, interactions and associations of the entities. By embedding ML models as predicates, GCRs unify ML prediction and logic reasoning to catch and fix errors, and explain ML prediction in logic.

Below are GCRs discovered from BioGRID [1], a real-life protein-protein interaction network, with the same pattern Q in Figure 1.

(a) GCR $\varphi_1 = Q[x_0, y_0](\mathcal{M}_1(x_0, x_1) > \delta_1 \land X_1 \rightarrow x_4.\text{val} = \text{physical})$, where $\mathcal{M}_1$ is a GCN model that predicts Interaction x_0's type by examining its features and post-translational modifications, and $X_1 = \langle x_2.\text{id} = y_0.\text{id} \land y_2.\text{val} = \text{virus}\rangle$. Intuitively, φ_1 corrects the false-positive predictions of $\mathcal{M}_1$ if (1) x_2 and y_0 are the same interactor, and (2) y_0 originates from an organism identified as a virus.

(b) GCR $\varphi_2 = Q[x_0, y_0](x_4.\text{val} = \text{genetic} \land X_2 \rightarrow \mathcal{M}_2(x_0, x_1) > \delta_2)$, where $\mathcal{M}_2$ is a GCN model for interaction throughput prediction, and $X_2 = \langle x_2.\text{id} = y_0.\text{id} \land y_2.\text{val} = \text{coli}\rangle$. It indicates that $\mathcal{M}_2$ predicts x_0 as a low-throughput interaction because (1) x_0 is a genetic interaction and (2) x_2 is identical to Interactor y_0, a derivative of coli.

Semantics. For GCR $\varphi = Q[x_0, y_0](X \rightarrow p_0)$, a *match* of Q in a graph G is a homomorphic mapping of pattern nodes in Q to nodes in G. Match h *satisfies* a predicate p of Q, denoted by $h \models p$, if (a)

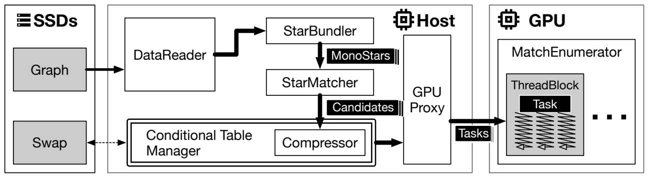

Figure 2: The pipelined architecture of MiniClean.

when p is $x.A \oplus y.B$, $h(x).A \oplus h(y).B$, similarly for $z.A \oplus c$; and (b) when p is $\mathcal{M}(x.\bar{A}, y.\bar{y})$, $\mathcal{M}$ predicts true at $(h(x).\bar{A}, h(y).\bar{B})$.

A *violation* of φ is a match h such that $h \models p$ for each predicate in X, but $h \not\models p_0$. Intuitively, each violation represents an error, *i.e.,* a duplicate (*e.g.,* $h(x).\text{id} = h(y).\text{id}$) or a conflict (*e.g.,* $h(x).A \neq h(y).B$).

Graph cleaning with GCRs. There are three major tasks.

Rule discovery. Given graph G, this task is to mine a cover Σ_c of GCRs from (samples) of G. It discovers a set Σ of GCRs, each meeting predefined thresholds σ and δ for support and confidence, respectively. It returns the cover Σ_c of Σ, which is a minimum set of non-redundant GCRs that is "equivalent to" Σ, *i.e.,* each GCR in Σ can be entailed by Σ_c via logic implication, and vice versa.

Error detection. This task is to catch and return all violations of the GCRs as errors, by applying the mined GCRs of Σ to graph G.

Error correction. To fix duplicates and conflicts in G, this task recursively applies GCRs of Σ to G whenever possible, fixing all errors, *e.g.,* if h is a violation of φ and p_0 is $x.A = y.B$, it enforces $h(x).A = h(y).B$ in G (see [5] for details). It also accumulates validated facts $h(x).A = h(y).B$ in a set Γ, and references Γ in the process. The process is Church-Rosser, *i.e.,* it converges at the same fixes regardless of the rules in Σ used and their application order. The output is a corrected graph G with all detected errors resolved.

By star patterns and conditions (1)-(2) in the definition GCRs, error detection and correction are in PTIME with GCRs [5].

Algorithms. The key to all graph cleaning tasks is the enumeration of all candidate matches of pattern Q in GCR $\varphi = Q[x_0, y_0](X \rightarrow p_0)$; it involves pattern matching and is computationally expensive.

Leveraging star patterns, a PTIME parallel algorithm $\mathcal{A}$ is in place [5] for match enumeration. Suppose that dual pattern Q is composed of paths $p_1, \ldots, p_k$ from a center (x_0 or y_0) to each leaf. Algorithm $\mathcal{A}$ enumerates matches of each path p_i, producing candidate pairs of path matches by dynamic programming. It filters candidates in the process, imposing relevant unary predicates as constraints. Then, by assembling the path matches at the centers x_0 and y_0, respectively, and checking the associated predicates, it finds all matches of pattern Q, in $O((|Q| + |X|)^2 |G|^2)$ time.

2.2 Workflow and Architecture

Algorithm $\mathcal{A}$ of [5] is designed for multi-machine systems, which partition G into fragments, load the fragments to memory of different machines at once, and enumerate matches with different machines in parallel. However, a typical single machine lacks the resources. On a single machine with GPUs, $\mathcal{A}$ does not work well due to (a) irregular data access for graph structures; (b) repetitive computations for common sub-patterns; (c) heavy memory footprint for intermediate results; and (d) limited GPU parallelism.

In light of these, MiniClean proposes a two-stage workflow to enumerate matches for a set of GCRs, with a single machine.

Stage 1: Star matching. Given a GCR $\varphi = Q[x_0, y_0](X \rightarrow p_0)$ and a graph G, this stage decomposes φ into *mono-star components* $Q_x[x_0, \bar{x}](X_1)$ and $Q_y[y_0, \bar{y}](X_2)$, where X_1 (resp. X_2) preserves the unary predicates from X associated with Q_x (resp. Q_y). It groups similar mono-star components of a set Σ of GCRs into bundles and matches them using CPUs. These matches, referred to as *star matches*, are stored in succinct tables (see Section 2.3 for details).

Stage 2: Cross-star enumeration. For each GCR φ in Σ, denote by C_x and C_y the matches of its mono-star components in succinct tables, from stage 1. Stage 2 scans each pair in the Cartesian product $C_x \times C_y$ on GPUs to verify if it satisfies condition X of φ.

This workflow exploits the strengths of CPU and GPU for different stages. By executing star matching on CPU, it efficiently handles irregular graph data and large sets of star matches, leveraging the access to larger memory and external storage. The cross-star enumeration is offloaded to the GPU, which excels at highly parallel tasks.

As shown in Figure 2, MiniClean adopts a pipeline to orchestrate the two-stage workflow for match enumeration.

2.3 Host-Side Optimizations

MiniClean speeds up star matching (Stage 1) as follows.

Bundled processing. MiniClean bundles "similar" mono-star components and matches them collectively to reduce repetitive computations. As common substructures are often shared by similar stars, intermediate data can be reused during the enumeration. Specifically, MiniClean bundles two mono-star components $Q_1[x_0](X_1)$ and $Q_2[x_0](X_2)$ if their center has the same label and the two stars share at least one common path. The *bundle* of $Q_1[x_0](X_1)$ and $Q_2[x_0](X_2)$ becomes a new mono-star bundle $Q_{1+2}[x_0](X_1 \cup X_2)$, where the combined star Q_{1+2} is the join of Q_1 and Q_2 at the common center x_0, with the shared paths deduplicated.

Compared to the independent matching of two mono-stars, the bundled processing (1) can reduce the overall computational workload by a factor of $\Theta(\frac{|Q_{1*2}|+|X_1 \cap X_2|}{|Q_1|+|Q_2|+|X_1|+|X_2|})$. Moreover, it produces a single shared set of matches for the mono-star bundle, reducing both intermediate results and CPU-GPU data transfers.

Star match compression. Matching a mono-star bundle generates a large number of matches of star patterns. To reduce intermediate results, we propose a succinct table for star matches, compressing a long table losslessly. The *succinct table* organizes the star matches around the central node of the star pattern: (1) each row is indexed by a unique vertex v in G, where v is a match of the center x_0 of the star pattern. In other words, a row keyed by v encodes all star matches centered at v. (2) In a row keyed by v, each column encodes a complete list of the matches of a specific leaf. The columns only include matches of leaves for cross-star checking in Stage 2.

A complication is that we must differentiate the star matches for each individual star in the bundle. Adapting conditional tables, we extend the succinct table with condition columns. This structure enables access to relevant matches without recomputing joins, facilitating fast recovery and access to star matches.

2.4 Hybrid Parallel Model

MiniClean exploits multiple parallel modes and employs an adaptive schedule to maximize pipeline throughput and performance.

Table 1: Efficiency and accuracy of MiniClean vs. ML models.

System	BioGRID			SemScholar		
	Time	ER-F1(%)	CR-F1(%)	Time	ER-F1(%)	CR-F1(%)
MiniClean	312.7s	97.5	97.2	4089.1s	94.6	70.6
Ditto	7.7×	91.0	N/A	4.3×	90.3	N/A
KGClean	11.9×	N/A	54.9	11.9×	N/A	29.8

Table 2: Performance evaluation: match enumeration.

Single-machine (BioGRID)		Scalability (SemScholar)		Ablation study (SemScholar)	
System	**Time (s)**	**System**	**Time (s)**	**System**	**Time (s)**
Blaze	OOM	GCRClean-32-Node	8.09×	noBundle	1.35×
MiniGraph	65.34×	MiniClean-2-GPU	0.64×	noPipelinedPar	1.26×
Hyperblocker	11.92×	MiniClean-4-GPU	0.48×	noIndPar	1.12×
MiniClean-1-GPU	259.6s	MiniClean-1-GPU	4251.08s	MiniClean-2-GPU	2993.7s

Multi-mode parallelism. MiniClean uses three types of parallelism to fully utilize the parallel processing power of CPU and GPU.

Pipelined parallelism. MiniClean pipelines the two stages of match enumeration across CPU and GPU, overlapping them to improve throughput and mitigate the cost of SSD data swapping and CPU-GPU data transfers. Unlike prior GPU-accelerated systems that offload nearly all computation to the GPU and use the CPU only for lightweight tasks, MiniClean exploits the strengths of both.

SIMD parallelism. MiniClean leverages SIMD parallelism to process different star matches concurrently with multiple GPU threads. The regular structure of the succinct table facilitates efficient SIMD parallelization, ensuring optimal GPU resource utilization.

Independent parallelism. MiniClean runs multiple cross-star enumeration tasks on the GPU simultaneously via independent parallelism, each processing a pair of star matches buckets. This allows multiple tasks to share the GPU resources to improve utilization.

3 Demonstration Overview

This section presents our demonstration setting and plan[1].

3.1 Demonstration Setting

Graphs. We use two real-life graphs: (a) BioGRID [1], a PPI network with 4M nodes and 26.5M edges (Section 2.1); and (b) SemScholar [2], a citation graph with 0.16B nodes and 0.75B edges.

Baselines. We will evaluate (1) accuracy against ML solutions Ditto [9] for ER and KGClean [3] for CR, both trained using their default settings; and (2) efficiency of graph cleaning versus (a) single machine out-of-core Blaze [11] and hybrid system MiniGraph [8], and (b) multi-machine graph cleaning systems GCRClean [5].

Testbeds. MiniClean is powered by 2× Intel Xeon Gold 6254 @3.10GHz CPUs, each having 16 physical cores with hyperthreading. It is also equipped with 64 GB of DDR4 RAM , 4× NVIDIA Tesla V100 32GB GPUs, and 4× 2TB Samsung 990Pro NVMe SSDs.

We will deploy multi-machine GCRClean on a GraphScope [4] cluster with up to 32 machines, where each is powered by an Intel Xeon @2.2GHz 12-core CPU and 128GB DDR4 RAM.

3.2 Demonstration Plan

We will invite participants to experience the following.

(1) Graph cleaning. We will walk users through the following.

Rule discovery. Users may select an input graph and configure parameters for rule discovery (support, confidence and pattern size). MiniClean will then discover GCRs and visualize them in a rule discovery panel, where users can inspect each rule. Two example GCRs (φ_1 and φ_2) discovered from BioGRID are showcased in Section 2.1.

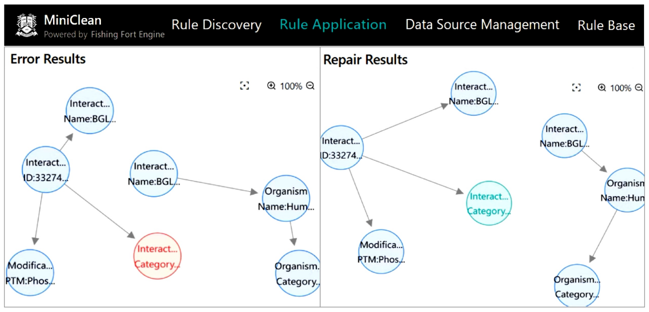

Figure 3: A snapshot of the graph cleaning panel.

Error detection and correction. Using the mined rules, one may opt to detect or correct errors in graphs using a system panel. MiniClean executes the selected task and visualizes the errors caught or fixed, together with the corresponding GCR. Errors are highlighted in red and their corrections in green (see Figure 3). Users can inspect each fix to errors via an interactive interface.

(2) Performance. Users are invited to try out different combinations of optimization toggles. Statistics are shown in Tables 1 and 2.

Overall performance. The F1-score of MiniClean is >94.6% for ER and >70.6% for CR, outperforming Ditto by 4.3%–6.5% and KGClean by 40.8%–42.3%. For efficiency, MiniClean is 65.34× faster than CPU-based MiniGraph, 11.92× faster than GPU-accelerated Hyperblocker, and 8.09× faster 32-node GCRClean. Using 2 (resp. 4) GPUs, MiniClean reduces execution time to 64% (resp. 48%).

Impact of bundled processing. MiniClean achieves a 1.35× speedup over noBundle, which processes each mono-star component separately. This verifies that the bundled processing reduces the duplicated computation and data transfers, and better utilizes GPU.

Impact of hybrid parallelism. MiniClean is 1.26× and 1.12× faster than noPipelinedPar and noIndPar, which turn off pipelined and independent parallelism, respectively. Thus hybrid parallelism overlaps CPU/GPU and I/O operations, and better uitlizes resources.

References

[1] 2024. The BioGRID Database. *https://thebiogrid.org/*.
[2] 2024. SemanticScholar Academic Graph. *https://www.semanticscholar.org/*.
[3] Congcong Ge et al. 2020. KGClean: An Embedding Powered Knowledge Graph Cleaning Framework. *CoRR* abs/2004.14478 (2020).
[4] Wenfei Fan et al. 2021. GraphScope: A Unified Engine For Big Graph Processing. *PVLDB* 14, 12 (2021), 2879–2892.
[5] Wenfei Fan et al. 2025. Making It Tractable to to Detect and Correct Errors in Graphs. *ACM Trans. on Database Systems* (2025), 16:1–16:75.
[6] Xiangxiang Zeng et al. 2022. Toward better drug discovery with knowledge graph. *Current opinion in structural biology* 72 (2022), 114–126.
[7] Xiangyu Zhi et al. 2023. CoroGraph: Bridging Cache Efficiency and Work Efficiency for Graph Algorithm Execution. *PVLDB* 17, 4 (2023), 891–903.
[8] Xiaoke Zhu et al. 2023. MiniGraph: Querying Big Graphs with a Single Machine. *PVLDB* 16, 9 (2023), 2172–2185.
[9] Yuliang Li et al. 2020. Deep Entity Matching with Pre-Trained Language Models. *PVLDB* 14, 1 (2020), 50–60.
[10] Wenfei Fan and Shuhao Liu. 2024. Fishing Fort: A System for Graph Analytics with ML Prediction and Logic Deduction. In *The Provenance of Elegance in Computation - Essays Dedicated to Val Tannen*. Schloss Dagstuhl, 6:1–6:18.
[11] Juno Kim and Steven Swanson. 2022. Blaze: Fast Graph Processing on Fast SSDs. In *SC*. 44:1–44:15.

[1]A companion video is available at https://www.bilibili.com/BV141F8e9E6S.