ORB: Empowering Graph Queries through Inference

Sonia Horchidan, Paris Carbone

KTH Royal Institute of Technology, Stockholm, Sweden

Abstract

Executing queries on incomplete, sparse knowledge graphs yields incomplete results, especially when it comes to queries involving traversals. In this paper, we question the applicability of all known architectures for incomplete knowledge bases and propose ORB: a clear departure from existing system designs, relying on Machine Learning-based operators to provide inferred query results. At the same time, ORB addresses peculiarities inherent to knowledge graphs, such as schema evolution, dynamism, scalability, as well as high query complexity via the use of embedding-driven inference. Through ORB, we stress that approximating complex processing tasks is not only desirable but also imperative for knowledge graphs.

1. Introduction

The data management community has shown particular interest in systems that can sustain graph-like data and, particularly, knowledge graphs (KGs) due to their intricate properties and structure that impose significant challenges. As a result, multiple classes of graph management systems exist nowadays, optimizing for various goals: (1) graph databases excel in OLTP workloads, (2) graph processing systems handle analytical-type queries at scale, and (3) graph streaming systems maintain rolling aggregates and statistics over evolving graphs. Recent analyses conclude that no system can tackle all the challenges of massive, dynamic, and heterogeneous graphs [1]. However, most importantly, field practitioners often overlook the elephant in the room: any analysis on large graphs is bound to be incomplete due to the fundamental tendency toward sparsity in naturally interconnected data [2].

Circumventing incompleteness has instead been the target of the Machine Learning (ML) research community. Recent innovative results are based on the inference power of Graph Neural Networks and their respective applications to KGs [3, 4]. Despite their impressive achievements, ML-driven methods still lack fundamental features that prevent them from being integrated into scalable query-serving systems, such as technical limitations in the available infrastructure or a general lack of trust in black-box models.

In this paper, we propose ORB, our vision architecture of an ML-driven graph database that allows for querying incomplete knowledge bases with novel insights and conclusions. At the same time, ORB acknowledges the benefits of offering inferred results and attempts to address crucial problems in graph data management, such as query complexity or data scalability, through the use of inference-based operators and user-defined uncertainty thresholds. With ORB, we advocate for a shift towards embracing approximate query results for deep analytics

1st International Workshop on Data Management for Knowledge Graphs (DMKG 2023) co-located with ESWC 2023, May 29th, 2023, Hersonissos, Greece

sfhor@kth.se (S. Horchidan); parisc@kth.se (P. Carbone)

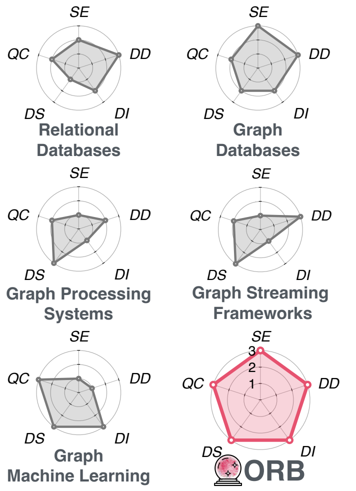

Figure 1: Systems coverage for the key challenges of KGs.

SC	SL	Definition
SE	1	The schema cannot change.
	2	Requires additional operations to support schema changes.
	3	The schema can change with no additional operations.
DD	1	Can only accommodate new entries via full reconstruction.
	2	Ingests new data entries in batches.
	3	Ingests new data on-the-fly.
DI	1	Cannot infer new knowledge.
	2	Manual inference logic possible using a query language.
	3	Can natively infer new knowledge.
DS	1	Can scale to non-skewed graphs.
	2	Can scale to skewed graphs using user-defined partitioning.
	3	Can scale to any graph size and skew automatically.
QC	1	Can efficiently process local queries.
	2	Can efficiently compute global iterative queries.
	3	Can efficiently compute nested queries.

Table 1: Key graph data management challenges. *SC* stands for Systems Challenge. *SL* stands for Support Level, which is defined as it follows: no support (1), limited support (2), full support (3).

on KGs; employing approximation is not a design option but a necessity in overcoming a series of critical shortcomings in graph reasoning today.

2. Key Challenges

To elaborate further on the complex dimensions surrounding KGs and inspired by the work of Lissandrini et al. [1], we compose a framework of properties (depicted in Table 1) that reflect core needs in modern query-serving systems.

Schema evolution (SE) measures the adaptability of a system to structural changes in the data. Knowledge graphs lack data schematization, allowing data of any type to be ingested ad infinitum. On the other hand, *data dynamism (DD)* defines the system's capability to enable evolution in terms of additions or updates that comply with the current schema. *Data incompleteness (DI)* involves answering questions given partial knowledge, where, naturally, the outcome is also bound to be incomplete. This problem is especially daunting for traversal-based computations, where the missing links strip away potential answers with each intermediate hop. Dealing with incompleteness implies the need to find novel methods for deriving new conclusions based on stored historical information. *Data scalability (DS)* measures the ability to scale to any graph size and skew. Even though it has been thoroughly researched for decades, scaling out to massive, power-law graphs that are at the same time, dynamic is still an open challenge [5]. Choosing an unsuitable partitioning algorithm can lead to increased query latency, especially for data traversals that combine information across partitions. Finally, *query complexity (QC)* refers to algorithmic computational complexity. For this framework, we define three different support levels as follows: (1) local queries that consist of point look-ups or local traversals, (2) global fixed-point iterative queries that compute a graph property in polynomial time (e.g., PageRank), and (3) nested queries that refer to intractable problems (such as counting motifs and multi-hop reasoning). The latter currently lacks effective optimization support in most modern systems, despite their critical need in the graph community [6].

ORB targets the data incompleteness challenge but also pursues the other dimensions through the lens of approximations via inference-based computations.

3. Overview of Existing Systems

We include a short overview of the state-of-the-art graph data management systems and discuss their coverage with respect to the core challenges identified in the previous section (Figure 1).

Relational Databases. The relational model requires manual interventions in the shape of expensive table alterations when ingesting out-of-schema data points. Hence, relational DBs offer partial *SE* support in our taxonomy. On the other hand, the relational model allows new data points to be inserted into the tables on-the-fly, thus fully supporting *DD*. As of today, no RDBMS can infer new conclusions automatically, but the capability to write custom inference logic in SQL is possible. For this reason, RDBMSes offer limited support only for *DI*. Next, a RDBMS can natively scale out but does not offer distribution-aware data partitioning, meaning that relational DBs offer no support for *DS*. Lastly, worst-case optimal join algorithms for subgraph queries on relational underlying data representation have shown potential in offering limited support for *QC* in our taxonomy [7, 8, 9].

Graph Databases. Graph DBs (e.g., Neo4j [10]) offer better support for *SE* than their relational counterpart through the adoption of the property graph data model [11]. The model is flexible enough to fully support *DD*. Regarding *DI*, the inference capability is partially ensured through the graph-tailored query language. Most native graph DBs currently do not offer automatic data sharding or do so by relying on naive hash partitioning, with the exception of Neo4j, which supports manual data partitioning only. For this reason, graph DBs offer only limited support for *DS*. Lastly, when it comes to *QC*, some specialized tools can efficiently compute global iterative queries (e.g., Neo4j's GDS library). However, nested graph problems have been largely overlooked due to their computational unfeasibility [1]. Thus, graph DBs offer limited support for *QC*.

Graph Processing Systems. Graph Processing Systems (GPSes) such as Pregel [12] and GraphLab [13], are data-intensive graph system libraries built on MapReduce-based frameworks. Their focus is computation at scale on stored, static data, which requires the schema to be known at compile time. Therefore, they are incapable of dealing with *SE*. In some cases, updates can occur in batches, making GPSes partially capable of *DD*. Next, GPSes can be programmed using simple, limited APIs such as the vertex-centric BSP model [12]. Using such APIs, the user can compose graph-global measurements that activate per existing vertex or neighborhood. However, this level of simplicity disallows computation on non-existing inferred elements of a graph; therefore, no support for *DI* is feasible. On the other hand, GPSes are purposefully designed to scale out to massive, skewed graphs, qualifying them for full support level for *DS* [14]. Lastly, they can handle global iterative queries but cannot efficiently compute nested queries, thus having limited support for *QC* in our framework.

Graph Streaming Frameworks. Graph Streaming Frameworks (GSFs) target high ingestion rates and compute global aggregates over streaming data. Similarly to GPSes, GSFs (1) require a-priori knowledge of the schema, resulting in limited support for *SE*, (2) provide restricted

Class	Query	Approximations
Local	Property Retrieval/ Link Retrieval/ Node Similarity	Grover et al. 2016 [15], Hamilton et al. 2017 [16], Galkin et al. 2022 [17]
Global Iterative	Node Importance	Park et al. 2019 [18]
	Distance	Merchant et al. 2022 [19]
	Clustering	Bianchi et al. 2020 [20]
Nested	Multi-Hop	Ren et al. 2020 [21]
	Motifs Counting	Ying et al. 2020 [22]
	Partitioning	Zwolak et al. 2022 [23]

Table 2: Examples of graph queries that can be approximated using graph ML techniques.

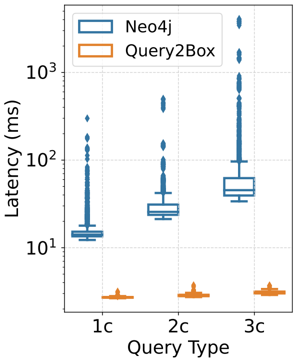

Figure 2: Multi-hop query latency comparison. 2c stands for two-hop queries.

APIs, having no support for *DI*, (3) are designed to scale out and have full support for *DS*, and (4) cannot compute nested workloads efficiently, thus supporting *QC* only partially. GSFs differ from GPSes in terms of *DD* support: GSFs are designed to effectively ingest new data in a streaming fashion.

This analysis concludes that no current system architecture is equipped to deal with incomplete knowledge bases, as their querying capabilities are built upon data traversals. Furthermore, challenges such as data scalability and query complexity still remain critical open problems.

4. Graph Machine Learning: The Promise

The emergent need for a sustainable solution to graph computing is driving researchers to seek a paradigm shift toward methodologies tolerating increasing levels of data incompleteness while avoiding the penalty of expensive graph traversals and computations. This section discusses the properties of graph ML, a promising direction to overcome these challenges.

Graph Representation Learning, specifically Graph Neural Networks (GNNs) [24], have been showing outstanding results in discovering unknown patterns and predicting insightful facts. Their innovation stems from the idea of translating the high-dimensional, non-euclidean space of graphs into latent, low-dimensional spaces. Each graph node, edge, or even subgraph can be transformed into a continuous, dense vector (i.e., *embedding*) that condenses both intrinsic features and adjacent topological information. Computations can then, in turn, be translated from non-euclidean spaces to the Euclidean latent spaces of embeddings.

The known capabilities of graph ML currently go beyond classifications or regressions, obtaining impressive estimates for intricate problems and alleviating the computational cost of running complex queries. Recent work hints at the capabilities of GNNs to approximate dynamic programming methods and solve any graph algorithm [25, 26, 27]. Table 2 shows examples of graph ML methods proposed in the literature to solve and approximate a selection of graph queries with remarkable inference results. With ORB, we acknowledge the potential of graph ML to enhance the capabilities of query-serving systems. To prove the outstanding benefits of graph ML in this context, we dedicate the following section to Query2Box [21], an embedding-based query execution method on KGs.

4.1. The Case of Query2Box

Query2Box [21] approximates first-order, multi-hop queries and offers accurate, predicted answers. Moreover, it computes the results in *constant time* (i.e., time complexity does not depend on the graph size), thus eliminating the need for expensive traversals on explicitly stored data. Query processing boils down to performing box projections and intersections in the latent space, followed by a cheap look-up on raw data to retrieve the results' explicit information [28]. Therefore, Query2Box caters to missing data in the knowledge bases with the additional advantage of accelerating multi-hop queries.

The source paper of Query2Box stands as a testimony to the practice of applying inference in extracting insightful answers even in incomplete data scenarios [21]. We now conduct a set of preliminary experiments to also quantify its expected query acceleration benefits in comparison to systems relying on traversal-based methods, such as Neo4j [1] . We choose multi-hop queries of different lengths and measure individual query latencies, which we define as the time it takes the system to process and return the results of a specific query. Rather than absolute values, we observe the trends and behavior of the two alternatives. The results (Figure 2) show that Query2Box answers reasoning queries in constant, reliable time, regardless of the cardinality of the result set and despite the non-optimized Python implementation. The query latency grows linearly with the query depth and has very low variation across the query spectrum. Whereas, Neo4j's query latency grows exponentially and results in a high number of outliers, hinting at an unpredictable execution time that depends on the degree of the nodes touched during the traversal.

4.2. Tackling Key System Challenges

Compared to the considered graph-oriented systems (Section 3), the inference capabilities of graph ML make it the only viable candidate to conquer *DI*. Furthermore, as shown in the previous section, a learned model has the potential to approximate nested, non-polynomial queries in polynomial time, proving that ML can achieve full support for *QC*. Regarding *DS*, we argue that embedding-driven query-answering does not depend on data locality because traversing raw data is replaced with traversing latent spaces. Still, *DD* and *SE* are not trivial to achieve. Most ML methods are inherently transductive; the ingestion of new data points is impossible after model deployment. Furthermore, inferring using out-of-schema data is unsuccessful since learned models usually require complete knowledge of all the data types at training time.

4.3. What about Uncertainty?

Relaxing the preciseness of graph queries to achieve fast answers comes at the cost of increased mistrust, which is especially problematic in this case since KGs model sensitive domains (e.g.,

[1] The VM was equipped with `Intel(R) Xeon(R) @2.00GHz` 32-core CPU, 120GB RAM, `Nvidia Tesla T4` GPU, 8GB VRAM, `Python 3.8`. We compared `Neo4j Community 4.4.9` against the original Query2Box implementation[2]on the FB15K-237 dataset [29]. We use the first 5000 queries defined as training data in Query2Box for each query length. Only non-trivial queries whose result set's cardinality is higher than 1 were depicted.

[2] https://github.com/hyren/query2box

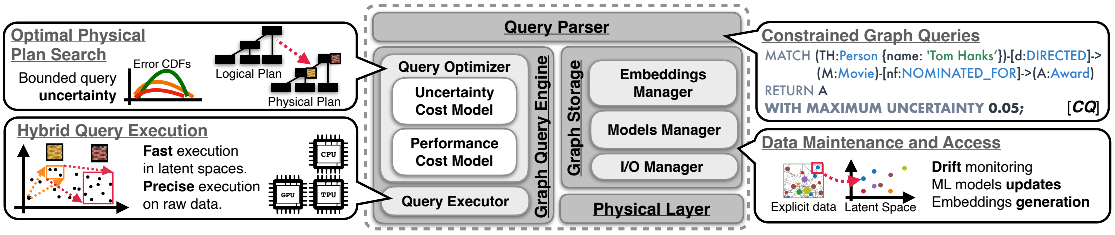

Figure 3: ORB architecture overview.

cybersecurity, autonomous vehicles, or medicine). The uncertainty of the predictions must be quantified on a per-query basis. To the best of our knowledge, no general-purpose framework exists to quantify the uncertainty of embedding-based approximations, but specialized uncertainty modeling exists for different methods (e.g., for embedding-based first-order logic queries [30, 31]). Conformal Prediction and Venn Predictors show two promising directions for increasing the trust in ML-based inferences by offering lightweight procedures for off-the-shelf to any black box model [32]. The first adapts the prediction to satisfy the user-defined error rate (i.e., significance level), while the latter outputs probabilistic predictions. We believe such methods can be adopted for graph ML tasks [33].

5. The ORB Architecture

Figure 3 depicts ORB, the visionary system architecture we designed based on the observations and conclusions presented in the previous sections. Contrary to traversal-based graph systems, ORB offers inferred results and moves expensive operations from the non-euclidean space of graph data to the latent spaces of learned embeddings. ORB's operators are designed hybridly: according to a user-defined uncertainty threshold, ORB chooses to execute operators either explicitly, on raw data, or in latent spaces, using ML inference. We now provide a high-level view of the key architecture components and weigh in on some essential design decisions.

5.1. Graph Query Engine

The graph query engine schedules, coordinates, and executes all graph queries. The design approach resembles relational DBs, starting with a logical plan obtained through query parsing and deriving an optimized physical plan with data/model access strategies. The graph engine comprises three modules: the query parser, the query optimizer, and the query executor. The query parser aims to extend existing graph DSLs. Similar to prior work [34], ORB annotates queries with user-defined uncertainty bounds, as it can be noted in the Cypher query *CQ* depicted in Figure 3. The bounds can assist the query optimizer in identifying the optimal physical plan. The query executor is reminiscent of traditional database query execution. We now dedicate the remainder of this section to the core of the ORB vision, the query optimizer.

Query Optimizer. In ORB, certain operators can be carried out in the latent spaces, resulting in a mix of raw data operations and embedding-based ones. The Query Optimizer lists all the candidate physical plans, considering that the same operator can be executed either on raw data or by potentially different ML models. If the user allows for approximate results, ORB

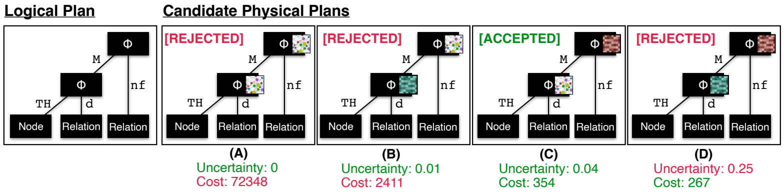

Figure 4: Simulated example of the cost-based plan optimization for the query CQ introduced in Figure 3.

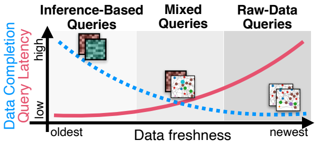

Figure 5: ORB's trade-off between completeness and latency.

must find the best trade-off between uncertainty and processing cost; running queries in the embedding space can lead to erroneous predictions but at a lower execution time. Thus, two cost models are required. A performance-based cost model assigns a score for each valid physical query plan based on the estimated processing cost. While the cost of traditional operators can be estimated using standard techniques, the execution of ML-based operators requires model inference calls that use specialized hardware acceleration. Therefore, estimating the cost of such operators boils down to a predefined number of matrix multiplications on respective hardware. An uncertainty-based cost model computes statistically sound prediction errors using lightweight instrumentation as suggested in Section 4.3. The uncertainty of a candidate plan can be computed by combining the uncertainty of each model used (e.g., as chained prior probabilities). If no plan can be identified to satisfy the required error threshold, ORB can choose to execute the query entirely on explicitly stored data, therefore providing zero uncertainty.

Example: CQ in Figure 3 illustrates a two-hop Cypher query that searches for all the award nominations for movies directed by Tom Hanks with a maximum allowed error of 5%. Figure 4 shows the corresponding logical plan. We define the operator $\phi(h, r)$ as the set of nodes t such that t is an answer to the path query $(h, r, ?)$. We assume that a link prediction model is available to the system. The optimizer can either choose to follow explicit links in the stored graph (depicted as ϕ⁙) or to rely on link prediction (shown as ϕ▦). The query optimizer identifies four candidate physical plans. Plan A is the cheapest physical plan with only traditional, explicit operators. Plans B and C contain a mix of embedding-based and raw data operators, while plan D operates entirely in the latent space. The cost optimizer identifies plan C as optimal for the given query since it shows the best performance cost at an acceptable uncertainty level.

5.2. Graph Storage

The storage component manages the access to persistent data. Given that ORB relies on embedding-based query executions, the system continuously maintains ML models. The I/O Manager operates similarly to today's systems, as its goal is to oversee the access at the raw data level. Distinctively, ORB extends the storage engine design by introducing managers for the models and embeddings, which constitute the main concern of this section.

Model Manager. One-off deployments of pre-trained ML models are unsuitable since the models become obsolete when the underlying distribution of the data changes [35]. The Model Manager incorporates change into the models. Two alternatives exist nowadays for keeping ML models up-to-date: (1) monitor the data and re-train the models when the drift is detected, or (2) integrate change into the models incrementally [36, 37, 38]. The first requires lower computational costs, at the expense of introducing non-determinism between answers of the same query asked at different times. Whereas, the latter happens as a background process that spends computation cycles continuously, but the embeddings can benefit from numerical stability over time [39].

Embedding Manager. The embeddings manager is tightly connected to the model manager; it is in charge of computing and updating the embeddings. The model type determines the embeddings generation process. Both transductive and inductive embedding methods [16] could benefit from caching policies for the embeddings of heavy hitters to alleviate either the I/O penalty of retrieving embeddings from persistent storage or the computational cost of computing new embeddings.

6. Design Prospects

We discuss ORB's design decisions and highlight the trade-offs, advantages, and limitations of replacing explicit query executions with ML-based approximations.

Benefits. Besides ”complete” results, the ORB architecture poses significant potential benefits:
1. Fast approximate answers. ORB's envisioned design avoids expensive and unnecessary computations that result in nonetheless incomplete results. We argue that, when it comes to KGs, computationally intensive, exact answers are not desirable since the data is notoriously incomplete. Thus, burning computation cycles for accurate answers is impractical.
2. Predictably low latency. ORB's performance cost model reliably the query execution time. The query plans will consist of a finite set of traditional database operators and ML model inferences. The cost of inference boils down to the number of pipelined matrix multiplication operations on respective hardware and is invariant to the input and intermediary results.
3. Seamless integration with modern hardware acceleration. Query answering is handled by ML scoring that can easily make use of modern hardware acceleration techniques, as opposed to traditional graph mining techniques that suffer from irregular access patterns [40].
4. Model-locality, not Data-locality. Contrary to traversal-based systems, data locality becomes less critical since there is no need to materialize all intermediate steps of a query, thus avoiding altogether pointer-chasing or maintaining indices for this purpose.
5. Raw Graph State Management. The graph storage layer can potentially be substituted by existing RDBMSs or graph DBs transparently to the rest of the system. The core requirements of this layer are (1) fast point lookups, and (2) support for model dynamic, heterogeneous data.

Latency versus Completeness. Graph ML cannot handle schema evolution efficiently due to models requiring prior knowledge of the schema. However, ORB executes query plans on a mix of explicitly stored data and embeddings. To achieve full *SE* and *DD*, ORB falls back on following raw data links for queries that operate on new data types while the models have time to be updated to incorporate the new information. The anticipated outcome is, therefore, a slower query execution with no new derived conclusions. As time passes and the models get updated, the queries can rely on inference-based calls and obtain low-latency responses with high incompleteness support. Under these conditions, we acknowledge the trade-off that emerges between query latency and data incompleteness support, which can be noted in Figure 5.

Open Problems and Opportunities. Even though ORB empowers graph exploration with promising benefits, we must acknowledge the open problems related to our visionary architecture and highlight the interesting research directions required to bring this vision to fruition.
Challenge #1. Inductive graph ML models. Graph ML should advance toward fully inductive models that can score unseen data points on-the-fly [16]. Full model inductiveness is set to offer desirable advantages in adaptability and throughput, leading to seamless integration in data management systems.
Challenge #2. Low-latency predictions. The (scarce) inductive graph ML methods that exist mostly rely on extracting k-hop node neighborhoods. Retrieving this information from the persistent storage of machines storing different graph partitions can quickly become a bottleneck [41].
Challenge #3. Inference and heterogeneity. Current graph ML methods offer little to no support for evolving schemas. Efforts such as one-shot or few-shot learning should be explored further to avoid expensive model re-training and adapt to new data types [42, 43].
Challenge #4. Uncertainty modeling. ORB requires error guarantees for ML-backed inferences that can be estimated at low computational expenses before the inference is executed so that the cost model can efficiently choose the appropriate physical plan.

Discussion Finally, we discuss the applicability of ORB to use cases of critical nature. ORB will not benefit situations and queries where precise answers or explicit graph traversals are essential. For this reason, ORB allows the users to constrain the error of the result; if a query does not allow for uncertain, predicted results, ORB will act as a traditional database. Therefore, ORB could potentially be built as an extension of existing databases.

7. Related work

Heavy research currently gravitates around ML algorithms, yet, an all-encompassing practical system approach is still in need. For example, we observe works focusing on training embedding models efficiently but failing to address continuous serving and monitoring [44, 45]. The time is ripe for such systems to emerge and potentially solve the major challenges of graph exploration.

Our work is a possible materialization of Neural Graph Databases, which was introduced in a recent report to address the incompleteness assumption of large KGs using Graph Representation Learning powered by ML inference [46]. Furthermore, similar research directions attempt to bring predictive capabilities to SPARQL via the usage of embeddings [47]. ORB extends these lines of work by aiming to incorporate error-guided query optimization techniques.

Secondly, the research area of neural databases makes use of Natural Language Processing to ingest and query unstructured data [48]. The ORB vision complements this line of work with ML-assisted query acceleration and result completion of common graph queries with no modifications to the database user interface.

A related area of work to ORB is approximate query processing (AQP) [34, 49, 50, 51, 52], which has seen research advancements in the use of learned models and indexes to improve performance. These new approaches can achieve faster and more accurate approximations of queries using a smaller amount of memory compared to traditional sample-based AQP [53, 54]. ORB is aligned with this trend by utilizing embedding-based models to approximate graph queries and also including mechanisms to estimate the approximation error.

Materialized views also relate to the ORB vision as another common methodology of "preparing" certain query results in advance [55, 56]. ORB's envisioned storage layer can be enhanced from view materialization for the storage of both raw data and produced embeddings.

Finally, ORB's objective is similar to that of knowledge graph systems like Vadalog [57], which provide logical reasoning capabilities. However, ORB aims to achieve reasoning through lightweight ML models with bounded state size, rather than computationally expensive rule-based methods.

8. Conclusion

We foresee that KGs will be the catalyst for the next-generation data-driven applications; therefore, designing systems well-equipped to handle KGs is essential. In this paper, we introduced ORB, our visionary system architecture that tackles data incompleteness. ORB opens new research avenues in graph processing and optimization, with interesting challenges relating to model-driven data management. We support that a proof-of-concept implementation of ORB could empower new opportunities in data analysis, granting users richer insights and better control over uncertainty with significantly lower overhead.

Acknowledgments

This work was supported by the Swedish Foundation for Strategic Research (BD15-0006), Wallenberg Foundation (WASP), and Google Cloud for Education. We also thank Vasiliki Kalavri for her valuable feedback.

References

[1] M. Lissandrini, D. Mottin, K. Hose, T. B. Pedersen, Knowledge graph exploration systems: are we lost?, in: CIDR, www.cidrdb.org, 2022.

[2] R. West, E. Gabrilovich, K. Murphy, S. Sun, R. Gupta, D. Lin, Knowledge base completion via search-based question answering, in: WWW, ACM, 2014, pp. 515–526.

[3] Q. Wang, Z. Mao, B. Wang, L. Guo, Knowledge graph embedding: A survey of approaches and applications, IEEE Trans. Knowl. Data Eng. 29 (2017) 2724–2743.

[4] Y. Zhou, H. Zheng, X. Huang, S. Hao, D. Li, J. Zhao, Graph neural networks: Taxonomy, advances, and trends, ACM Trans. Intell. Syst. Technol. 13 (2022) 15:1–15:54.

[5] Z. Abbas, V. Kalavri, P. Carbone, V. Vlassov, Streaming graph partitioning: an experimental study, Proceedings of the VLDB Endowment 11 (2018) 1590–1603.

[6] S. Sahu, A. Mhedhbi, S. Salihoglu, J. Lin, M. T. Özsu, The ubiquity of large graphs and surprising challenges of graph processing, Proc. VLDB Endow. 11 (2017) 420–431.

[7] X. Feng, G. Jin, Z. Chen, C. Liu, S. Salihoğlu, Kùzu graph database management system, in: CIDR, www.cidrdb.org, 2023.

[8] M. J. Freitag, M. Bandle, T. Schmidt, A. Kemper, T. Neumann, Adopting worst-case optimal joins in relational database systems, Proc. VLDB Endow. 13 (2020) 1891–1904.

[9] A. Mhedhbi, S. Salihoglu, Optimizing subgraph queries by combining binary and worst-case optimal joins, Proc. VLDB Endow. 12 (2019) 1692–1704.

[10] Neo4j, Inc., Neo4j, 2022. URL: https://neo4j.com/, [Online; accessed February 13, 2023].

[11] M. Lissandrini, M. Brugnara, Y. Velegrakis, Beyond macrobenchmarks: microbenchmark-based graph database evaluation, Proceedings of the VLDB Endowment 12 (2018) 390–403.

[12] G. Malewicz, M. H. Austern, A. J. C. Bik, J. C. Dehnert, I. Horn, N. Leiser, G. Czajkowski, Pregel: a system for large-scale graph processing, in: SIGMOD Conference, ACM, 2010, pp. 135–146.

[13] Y. Low, J. Gonzalez, A. Kyrola, D. Bickson, C. Guestrin, J. M. Hellerstein, Distributed graphlab: A framework for machine learning and data mining in the cloud, Proceedings of the VLDB Endowment 5 (2012).

[14] J. E. Gonzalez, Y. Low, H. Gu, D. Bickson, C. Guestrin, Powergraph: Distributed graph-parallel computation on natural graphs, in: OSDI, USENIX Association, 2012, pp. 17–30.

[15] A. Grover, J. Leskovec, node2vec: Scalable feature learning for networks, in: KDD, ACM, 2016, pp. 855–864.

[16] W. L. Hamilton, Z. Ying, J. Leskovec, Inductive representation learning on large graphs, in: NIPS, 2017, pp. 1024–1034.

[17] M. Galkin, E. G. Denis, J. Wu, W. L. Hamilton, Nodepiece: Compositional and parameter-efficient representations of large knowledge graphs, in: ICLR, OpenReview.net, 2022.

[18] N. Park, A. Kan, X. L. Dong, T. Zhao, C. Faloutsos, Estimating node importance in knowledge graphs using graph neural networks, in: KDD, ACM, 2019, pp. 596–606.

[19] A. Merchant, A. Gionis, M. Mathioudakis, Succinct graph representations as distance oracles: An experimental evaluation, Proc. VLDB Endow. 15 (2022) 2297–2306.

[20] F. M. Bianchi, D. Grattarola, C. Alippi, Spectral clustering with graph neural networks for graph pooling, in: ICML, volume 119 of *Proceedings of Machine Learning Research*, PMLR, 2020, pp. 874–883.

[21] H. Ren, W. Hu, J. Leskovec, Query2box: Reasoning over knowledge graphs in vector space using box embeddings, in: ICLR, OpenReview.net, 2020.

[22] R. Ying, Z. Lou, J. You, C. Wen, A. Canedo, J. Leskovec, Neural subgraph matching, CoRR abs/2007.03092 (2020).

[23] M. Zwolak, Z. Abbas, S. Horchidan, P. Carbone, V. Kalavri, *GCNSplit*: bounding the state of streaming graph partitioning, in: aiDM@SIGMOD, ACM, 2022, pp. 3:1–3:12.

[24] M. Gori, G. Monfardini, F. Scarselli, A new model for learning in graph domains, in: Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005., volume 2, IEEE, 2005, pp. 729–734.

[25] M. Fereydounian, H. Hassani, J. Dadashkarimi, A. Karbasi, The exact class of graph functions generated by graph neural networks, arXiv preprint arXiv:2202.08833 (2022).

[26] A. Dudzik, P. Velickovic, Graph neural networks are dynamic programmers, CoRR abs/2203.15544 (2022).

[27] K. Xu, J. Li, M. Zhang, S. S. Du, K. Kawarabayashi, S. Jegelka, What can neural networks reason about?, in: ICLR, OpenReview.net, 2020.

[28] P. Indyk, R. Motwani, Approximate nearest neighbors: Towards removing the curse of dimensionality, in: STOC, ACM, 1998, pp. 604–613.

[29] K. Toutanova, D. Chen, Observed versus latent features for knowledge base and text

inference, in: CVSC, Association for Computational Linguistics, 2015, pp. 57–66.

[30] H. Ren, J. Leskovec, Beta embeddings for multi-hop logical reasoning in knowledge graphs, NeurIPS 33 (2020) 19716–19726.

[31] X. Chen, M. Boratko, M. Chen, S. S. Dasgupta, X. L. Li, A. McCallum, Probabilistic box embeddings for uncertain knowledge graph reasoning, in: NAACL-HLT, Association for Computational Linguistics, 2021, pp. 882–893.

[32] V. Vovk, A. Gammerman, G. Shafer, Algorithmic learning in a random world, Springer Science & Business Media, 2005.

[33] R. F. Barber, E. J. Candes, A. Ramdas, R. J. Tibshirani, Conformal prediction beyond exchangeability, arXiv preprint arXiv:2202.13415 (2022).

[34] S. Agarwal, B. Mozafari, A. Panda, H. Milner, S. Madden, I. Stoica, Blinkdb: queries with bounded errors and bounded response times on very large data, in: EuroSys, ACM, 2013, pp. 29–42.

[35] A. Tsymbal, The problem of concept drift: definitions and related work, Computer Science Department, Trinity College Dublin 106 (2004) 58.

[36] J. Wang, G. Song, Y. Wu, L. Wang, Streaming graph neural networks via continual learning, in: CIKM, ACM, 2020, pp. 1515–1524.

[37] L. Galke, B. Franke, T. Zielke, A. Scherp, Lifelong learning of graph neural networks for open-world node classification, in: IJCNN, IEEE, 2021, pp. 1–8.

[38] M. Perini, G. Ramponi, P. Carbone, V. Kalavri, Learning on streaming graphs with experience replay, in: Proceedings of the 37th ACM/SIGAPP Symposium on Applied Computing, 2022, pp. 470–478.

[39] M. Xu, Understanding graph embedding methods and their applications, SIAM Rev. 63 (2021) 825–853.

[40] X. Shi, Z. Zheng, Y. Zhou, H. Jin, L. He, B. Liu, Q.-S. Hua, Graph processing on GPUs: A survey, ACM Computing Surveys (CSUR) 50 (2018) 1–35.

[41] H. Zhou, A. Srivastava, H. Zeng, R. Kannan, V. K. Prasanna, Accelerating large scale real-time GNN inference using channel pruning, Proc. VLDB Endow. 14 (2021) 1597–1605.

[42] W. Xiong, M. Yu, S. Chang, X. Guo, W. Y. Wang, One-shot relational learning for knowledge graphs, in: EMNLP, Association for Computational Linguistics, 2018, pp. 1980–1990.

[43] C. Zhang, H. Yao, C. Huang, M. Jiang, Z. Li, N. V. Chawla, Few-shot knowledge graph completion, in: AAAI, AAAI Press, 2020, pp. 3041–3048.

[44] H. Ren, H. Dai, B. Dai, X. Chen, D. Zhou, J. Leskovec, D. Schuurmans, SMORE: knowledge graph completion and multi-hop reasoning in massive knowledge graphs, in: KDD, ACM, 2022, pp. 1472–1482.

[45] N. Sheikh, X. Qin, B. Reinwald, C. Lei, Scaling knowledge graph embedding models for link prediction, in: EuroMLSys@EuroSys, ACM, 2022, pp. 87–94.

[46] H. Ren, M. Galkin, M. Cochez, Z. Zhu, J. Leskovec, Neural graph reasoning: Complex logical query answering meets graph databases, arXiv preprint arXiv:2303.14617 (2023).

[47] H. Kang, S. Hong, K. Lee, N. Park, S. Kwon, On integrating knowledge graph embedding into SPARQL query processing, in: ICWS, IEEE, 2018, pp. 371–374.

[48] J. Thorne, M. Yazdani, M. Saeidi, F. Silvestri, S. Riedel, A. Y. Levy, From natural language processing to neural databases, Proc. VLDB Endow. 14 (2021) 1033–1039.

[49] Y. Park, B. Mozafari, J. Sorenson, J. Wang, Verdictdb: Universalizing approximate query

processing, in: SIGMOD Conference, ACM, 2018, pp. 1461–1476.

[50] S. Chaudhuri, B. Ding, S. Kandula, Approximate query processing: No silver bullet, in: SIGMOD Conference, ACM, 2017, pp. 511–519.

[51] S. Agarwal, H. Milner, A. Kleiner, A. Talwalkar, M. I. Jordan, S. Madden, B. Mozafari, I. Stoica, Knowing when you're wrong: building fast and reliable approximate query processing systems, in: SIGMOD Conference, ACM, 2014, pp. 481–492.

[52] S. Kandula, A. Shanbhag, A. Vitorovic, M. Olma, R. Grandl, S. Chaudhuri, B. Ding, Quickr: Lazily approximating complex adhoc queries in bigdata clusters, in: SIGMOD Conference, ACM, 2016, pp. 631–646.

[53] S. Thirumuruganathan, S. Hasan, N. Koudas, G. Das, Approximate query processing for data exploration using deep generative models, in: ICDE, IEEE, 2020, pp. 1309–1320.

[54] Q. Ma, A. M. Shanghooshabad, M. Almasi, M. Kurmanji, P. Triantafillou, Learned approximate query processing: Make it light, accurate and fast., in: CIDR, 2021.

[55] A. Gupta, I. S. Mumick, Maintenance of materialized views: Problems, techniques, and applications, IEEE Data Eng. Bull. 18 (1995) 3–18.

[56] J. M. F. da Trindade, K. Karanasos, C. Curino, S. Madden, J. Shun, Kaskade: Graph views for efficient graph analytics, in: ICDE, IEEE, 2020, pp. 193–204.

[57] L. Bellomarini, E. Sallinger, G. Gottlob, The vadalog system: Datalog-based reasoning for knowledge graphs, Proc. VLDB Endow. 11 (2018) 975–987.