What's New in Neo4j Graph Data Science 2.3

February 2023

What Is Neo4j Graph Data Science?

Neo4j Graph Data Science is an analytics and modeling engine that uses the relationships in your data to improve predictions. It plugs into enterprise data ecosystems so you can get more data science projects into production quickly. Using pretuned graph algorithms, data scientists explore billions of data points in seconds to identify hidden connections and generate compelling visualizations that lead to better stakeholder decision making.

Areas of Investment

Neo4j offers the only Graph Data Science engine built for data scientists to improve their predictions and ML models, at scale, with seamless integration across the data stack. We continue to build on the momentum of our 2.0, 2.1, and 2.2 releases and focus on building the most comprehensive Graph Data Science solution on the market.

We're investing in four key areas:

1. **Make Better Predictions:** Build a proof of concept and go to production using any data source to discover what's important, what's unusual, and what's next.
2. **Integrate With Your Data Ecosystem:** Integrate Graph Data Science with the existing tools across your technology stack and data pipeline using native connectors.
3. **Built for Data Scientists:** Work in a familiar environment and quickly demonstrate practical business value.
4. **Production Ready: Trusted, Scalable, and Robust:** Deployment flexibility and options for moving models into production get more data science projects adopted.

What's New?

Highlights from this release include:

Maker Better Predictions

Features that enable you to build a proof of concept and go to production using any data source to discover what's important, what's unusual, and what's next. This includes:

The Knowledge Graph Embedding – HashGNN enables fast, scalable, and high-performing graph ML by efficiently generating embeddings on heterogeneous graphs. It resembles a Graph Neural Network (GNN) architecture but without the high computational cost and complexity of model training that often hinders GNNs.

Try out the Knowledge Graph Embedding – HashGNN using publicly available IMDB data on the Neo4j Graph Data Science GitHub.

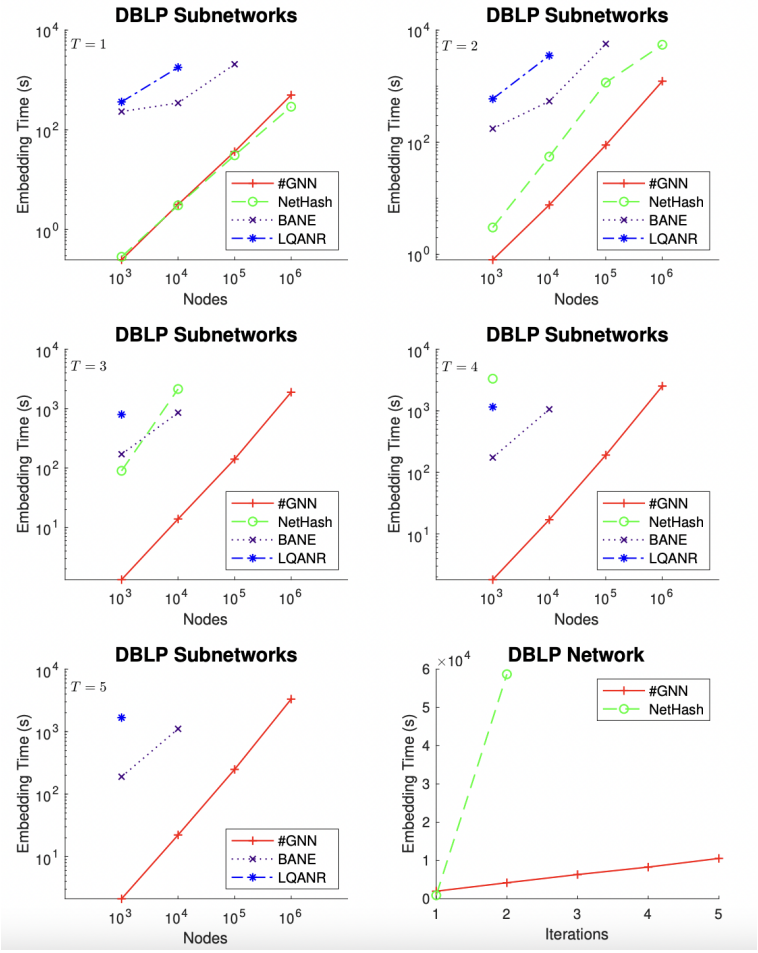

Figure 1: Scalability of the hashing-based algorithms on the DataBase systems and Logic Programming (DBLP) network – a citation dataset – and its four subnetworks.[1] The first five subplots show scalability with respect to the nodes of the subnetworks and the last one shows scalability on the original network. The subnetwork plots show the HashGNN achieves accuracy comparable to the learning-based algorithms, while running significantly faster than learning-based algorithms.

Improved Algorithms:

- **Leiden** is a community detection algorithm that uses hierarchical clustering to easily identify cohesive sub-communities within large networks. The algorithm separates nodes into disjoint communities to maximize a modularity score for each community. Modularity quantifies how densely connected nodes in a community are, compared to how connected they would be in a random network.

  Improvements include new parameters, progress tracking, and a new memory estimation mode.

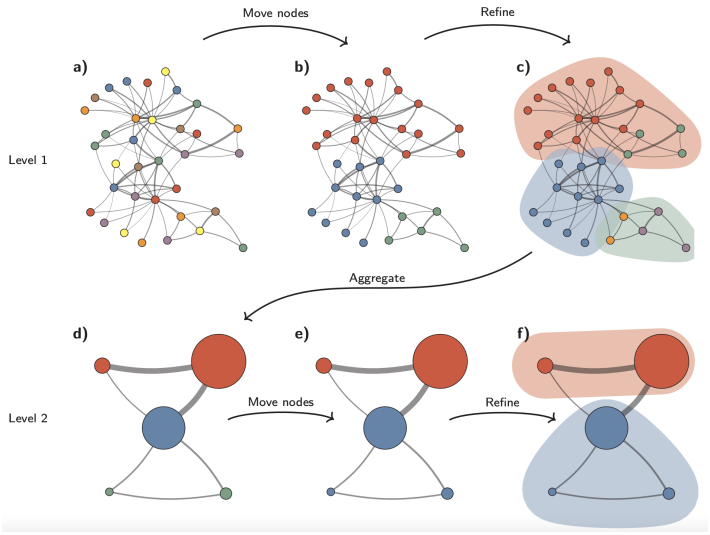

Figure 2: Leiden algorithm. The Leiden algorithm starts from a singleton partition (a). The algorithm moves individual nodes from one community to another to find a partition (b), which is then refined (c). An aggregate network (d) is created based on the refined partition, using the non-refined partition to create an initial partition for the aggregate network. For example, the red community in (b) is refined into two subcommunities in (c), which after aggregation become two separate nodes in (d), both belonging to the same community. The algorithm then moves individual nodes in the aggregate network (e). In this case, refinement does not change the partition (f). These steps are repeated until no further improvements can be made.[2]

- **Minimum Weight[3] Spanning Tree** is a pathfinding algorithm that is helpful for understanding which routes are available when there are multi-path solutions. It starts with a given node, finds all reachable nodes, and provides the set of relationships that connect the nodes together. The algorithm returns a spanning tree where the total weight of the relationships is minimized.

  Improvements include new supported modes, yield outputs, and new memory estimation modes.

New Algorithms

- **Minimum Directed[4] Steiner Tree** is a directed spanning tree that finds the shortest or least expensive route to a specific location from multiple starting locations. It does this by minimizing the sum of paths that exist from multiple source nodes to a single target node. The inverse is also possible.

New Link Prediction Parameter

- **Negative Relationships in Link Prediction:** Provide negative relationship examples for more options to train link prediction models and get them into production quickly. Previously link prediction models required data scientists to manually select negative examples (when a link does not exist between two nodes) as part of a random selection test. Now data scientists can train the model using their own negative relationship examples, enabling faster model training.

Integrate With Your Data Ecosystem

New and improved connectors, extensions, and integrations across the data pipeline ecosystem include:

Apache Arrow Integration for Graph Projections: Import and export massive graphs directly, at speeds of up to 8 million objects/second. Improvements include:

- **Incorporate Undirected Graphs:** Apache Arrow now supports undirected graph relationship types to automatically format a graph to be compatible with more algorithms in the platform and load more relationships faster and easier.

Built for Data Scientists

Work in a familiar environment and quickly demonstrate practical business value with:

Graph Data Science Python Client: Improvements to the Graph Data Science Python Client include:

- **Undirected relationship types**: Transform data for Graph Data Science faster, more efficiently, and with less code. Many algorithms benefit from an undirected graph which takes extra logic, know-how, and record duplication during construction. Users can automate this process to build or transform directed graphs into undirected graphs from the Graph Data Science Python Client.

- **Sample datasets:** Load out-of-the-box graph datasets like Cora and IMDB to get started quickly with rapid experimentation.

To learn more about the Graph Data Science Python Client updates, check out the changelog on GitHub.

Resources

Want to learn more about the 2.3 Graph Data Science release? Check out the new Graph Data Science Manual and Graph Data Science Changelog to learn more.

Glossary

1. From Wu, Wei, et al. “Hashing-Accelerated Graph Neural Networks for Link Prediction.” Proceedings of the Web Conference 2021, Apr. 2021. Crossref, https://doi.org/10.1145/3442381.3449884.

2. Traag, V.A., Waltman, L. & van Eck, N.J. “From Louvain to Leiden: guaranteeing well-connected communities.” Sci Rep 9, 5233 (2019). https://doi.org/10.1038/s41598-019-41695-z

3. **What does “weighted” mean?** It means the algorithm supports configuration to set node and/or relationship properties to use as weights. These values can represent cost, time, capacity, or some other domain-specific properties.

4. **What do “directed” and “undirected" mean?** “Directed” means the relationships (edges) of the graph flow in a specific direction. The direction is visually depicted as an arrow. “Undirected” means that the edges of the graph do not flow in a specific direction. An example of a directed relationship is Twitter. If you follow someone, that does not mean they follow you back. However, on Facebook, when you friend someone, the relationship is bi-directional and therefore, undirected.

Previous Release Announcements

Neo4j Graph Data Science 2.2 – October 2022

Neo4j Graph Data Science 2.1 – June 2022

Neo4j Graph Data Science 2.0 and AuraDS – April 2022