Provlepsis4j: Querying Future Graphs in Neo4j

Evangelos Iliadis
University of Ioannina
Ioannina, Greece
pcs00526@uoi.gr

Christos Gkartzios
University of Ioannina
Ioannina, Greece
chgartzios@cs.uoi.gr

Evaggelia Pitoura
University of Ioannina
Ioannina, Greece
Archimedes, Athena Research Center
Athens, Greece
pitoura@uoi.gr

ABSTRACT

Graphs model relationships in social, information, and collaboration networks, where users often need to reason about both the current graph and its possible evolution. However, graph databases such as Neo4j primarily query the present graph, while link-prediction methods typically run as separate workflows that output ranked candidate edges. We present Provlepsis4j, a Neo4j-based system that integrates link prediction with Cypher querying by maintaining both current and predicted graph states. Provlepsis4j offers an administrator view for configuring, running, and evaluating link-prediction models, and a user view for selecting among future timesteps, querying current and predicted graph states with the same Cypher queries, and inspecting answers side by side. In this way, Provlepsis4j makes predicted graph evolution directly queryable and helps users understand how predicted edges affect graph-query results.

PVLDB Reference Format:
Evangelos Iliadis, Christos Gkartzios, and Evaggelia Pitoura. Provlepsis4j: Querying Future Graphs in Neo4j. PVLDB, 19(12): 4770 - 4773, 2026.
doi:10.14778/3827998.3828118
PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/vagili/Provlepsis4j-Query-Future-Graphs.

1 INTRODUCTION

Graphs are widely used to represent entities and their interactions in social, information, and collaboration systems. Many queries ask how the graph will evolve, for example which users are likely to connect, which items will co-occur, or which nodes will become reachable through new paths. However, graph databases focus on querying the current graph, while predictive models are usually implemented as separate pipelines that output lists of edges.

This separation makes it difficult to combine prediction and querying in a single workflow. Running a link-prediction pipeline, exporting scores, and manually importing selected edges into the database requires engineering effort and discourages exploratory use. Queries that explicitly compare current and future graph states are particularly hard to support.

Proceedings of the VLDB Endowment, Vol. 19, No. 12 ISSN 2150-8097.
doi:10.14778/3827998.3828118

We address this gap with *Provlepsis4j*[1], a system that enables querying both the current and a predicted future graph in Neo4j[2] using the same Cypher queries. Provlepsis4j connects to Neo4j, stores the current and predicted future graphs, and provides a unified query interface for both. It integrates a link-prediction module that estimates edge-existence probabilities and materializes selected predictions in the predicted graph. Users can issue a Cypher query once and compare the answers on both graphs side by side. The system supports repeated prediction rounds, reports evaluation metrics for each prediction configuration, and includes heatmap views that summarize differences across configurations and against ground-truth edges.

To our knowledge, *Provlepsis4j* is the first system that supports querying both the current and a predicted future graph in Neo4j using the same Cypher queries. The system is inspired by our earlier work on future-time temporal path queries over evolving graphs [4], but here we focus on constructing predicted future graphs in Neo4j and querying them with Cypher.

In summary, this demo paper makes the following contributions:

- A Neo4j-based system that integrates link prediction with graph querying, allowing users to run Cypher queries on both current and predicted future graphs and compare answers side by side.
- A modular prediction pipeline that separates link-prediction models from embedding families, supports multiple prediction configurations, and extends the predicted graph through repeated timestamped prediction rounds.
- A web-based interface with administrator and user views, and a set of demonstration scenarios that guide users through loading graphs, configuring and evaluating prediction models, running predictions, comparing queries on current and predicted graphs, and visually analyzing prediction patterns and overlaps across prediction configurations.

2 SYSTEM DESCRIPTION

We structure Provlepsis4j around three core components, shown in Figure 1: a graph storage layer that utilizes the Neo4j graph database, a future graph prediction module, and a querying interface that compares current and predicted graphs. The web interface exposes two views: the *administrator view* which handles dataset loading, graph splitting, training and evaluating prediction models, and inspecting prediction overlaps. The *user view* focuses on extending the predicted future graph and issuing Cypher queries on the current and predicted graphs.

[1]Source code: https://github.com/vagili/Provlepsis4j-Query-Future-Graphs
[2]https://neo4j.com/

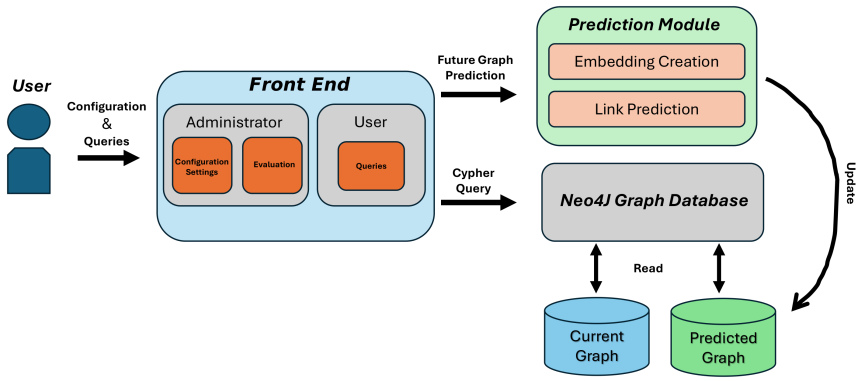

Figure 1: Architecture of Provlepsis4j.

2.1 Graph Storage in Neo4j

Neo4j serves as the storage backend for both the current graph and the predicted future graph states. Through the Provlepsis4j interface, users can connect to an existing Neo4j database or create a new graph from files. The resulting database is considered the *current graph* and acts as input to the prediction module.

Provlepsis4j supports two storage representations. In the separate-database representation, each predicted graph state is materialized as a separate Neo4j database that contains the observed relationships together with the selected predicted relationships, while the current graph remains unchanged. In the single-database multi-edge representation, observed and predicted relationships coexist in one database, and predicted edges carry metadata such as probability and prediction timestep. In both cases, the interface exposes current and predicted graph states via the same Cypher workflow.

2.2 Future Graph Prediction Module

The future graph prediction module takes as input a graph state and produces a set of candidate future edges. The module is organized around two choices: the link-prediction model and, when required, the embedding family used to represent nodes. This separation allows Provlepsis4j to support both regression-based predictors over graph embeddings and neural temporal models such as TGN [10].

Provlepsis4j supports both attribute-agnostic and attribute-aware embeddings. Attribute-agnostic methods such as Node2Vec [5] rely only on the graph structure. Attribute-aware methods such as FastRP [2], GraphSAGE [6], and HashGNN [13] can also incorporate node attributes when generating embeddings. TGN [10] also captures temporal information in evolving graphs.

For the selected prediction model, observed relationships provide positive examples, while sampled nonedges provide negative examples. The model is then trained to estimate an edge-existence probability for candidate node pairs. Regression-based models use node representations produced by the selected embedding family, while TGN learns a neural temporal scorer from the temporal interaction sequence. The administrator specifies the validation and test ratios in the *Split Graph* panel. Provlepsis4j uses these ratios to split the data, train the selected model, and report standard classification metrics on the test set.

Table 1: Runtime results on temporal datasets.

Email-Eu					
Nodes		**Edges**		**#Temp. Edges**	
986		16,064		327,333	
Metric	**Node2Vec**	**FastRP**	**GraphSAGE**	**HashGNN**	**TGN**
LP Training (s)	16.32	14.34	18.72	13.85	94.83
DB Creation (s)	49.32	48.15	47.23	46.79	48.33
CollegeMsg					
Nodes		**Edges**		**#Temp. Edges**	
1,893		13,835		59,791	
Metric	**Node2Vec**	**FastRP**	**GraphSAGE**	**HashGNN**	**TGN**
LP Training (s)	11.19	7.95	12.84	8.05	44.04
DB Creation (s)	61.83	62.00	62.27	62.41	62.92

For inference, Provlepsis4j scores a bounded candidate set of potential edges between node pairs that are not connected in the relevant graph state. The administrator configures the *Number of Predicted Edges* k and the *Edge Existence Probability Threshold* τ. The system ranks candidates by predicted probability and materializes up to k selected predictions, storing their probability and prediction-round timestamp. Repeating this process extends the predicted graph over multiple future timesteps, while already observed or previously predicted edges are excluded from later candidate sets. Table 1 reports dataset sizes, training time, and predicted-database creation time for the logistic-regression configurations on two temporal datasets. Runtime measurements were obtained using a single CPU core on an Intel Core i7-8700K 3.70GHz with 16GB RAM.

2.3 Query Interface

Provlepsis4j provides a unified query interface for both the current and the predicted future graphs. The interface has two views: an administrator view and a user view.

Figure 2: Administrator view of Provlepsis4j (*Scenario 2*).

Administrator view (Figure 2). The administrator view is used to configure and evaluate the prediction models. After loading a graph and defining train, validation, and test splits, the administrator selects a prediction model and, when needed, an embedding family. The main prediction parameters are the number of edges to predict and the probability threshold used during prediction. Internally, the system scores the candidate edges produced by the prediction module and orders them by predicted probability. The *Predicted edge inclusion* option controls whether later prediction rounds retrain the model using edges predicted in earlier rounds.

For each prediction configuration, the administrator view reports accuracy, precision, recall, F1, and AUC on the held out test set. It also includes a *Prediction Comparison* section on a fixed set of candidate edges. For each configuration, the system stores its predictions, computes their overlap with the ground truth edges and with the predictions of other configurations, and visualizes these overlaps as a heatmap. Each cell reports the fraction of edges predicted by both configurations. This helps evaluate which configurations agree or differ and how these patterns translate into changes in query answers on the predicted graphs.

User view (Figure 3). The user view focuses on querying the current and predicted future graphs. Users enter a Cypher query once and Provlepsis4j executes it on both. The results are shown side by side, so that differences between the two graphs are explicit. A color bar at the top of the view maps timesteps from *Present* to *Future*. Edges in the predicted graph are colored accordingly, from green for edges predicted at the earliest future timestep to yellow and red for edges predicted at later timesteps. When users hover over predicted edges, the interface also shows their prediction probability and timestamp. This allows users to see which parts of a query answer rely on earlier or later prediction rounds.

Query answers are presented both as tables and as interactive graph visualizations. This allows users to inspect which nodes and relationships satisfy a query in each graph and how their connectivity changes when predicted edges are taken into account. Typical uses include checking which new neighbors a node gains in the predicted graph or whether nodes that were previously disconnected become connected.

As additional prediction rounds are executed in the user view, the predicted graph is updated with new edges and timestamps. The *Current Predicted Timesteps* field shows how many future timesteps are currently present, while the *Additional Predicted Timesteps* control specifies how many new rounds to run when extending the predicted graph. The *Query Horizon* control lets users query the predicted graph up to a selected prediction timestep. The same queries can then be run again to observe how answers evolve over time and to compare current and predicted future graph states within a single querying environment.

3 DEMONSTRATION SCENARIOS

Our demonstration is organized around three scenarios that highlight Provlepsis4j from different angles: setting up graphs and prediction models, inspecting prediction patterns across different prediction configurations, and comparing query answers on current and predicted graphs.

Scenario 1: Setting Up Graphs and Models

The first scenario introduces the configuration steps needed to run Provlepsis4j. We begin by loading a graph, either by connecting to an existing Neo4j database or by importing a graph from files.

After the graph is available, we configure the prediction module. In the *Split Graph* panel the demonstrator sets the validation and test ratios, selects a prediction model and, when needed, an embedding family, and chooses the number of edges to predict and the edge-existence probability threshold. Provlepsis4j trains the selected model, reports standard evaluation metrics on a held out test set, scores a candidate set of nonedges, and materializes the selected predictions in the predicted graph. This scenario shows that the entire pipeline, from loading the dataset to populating the predicted graph, runs inside the web interface.

Scenario 2: Exploring Prediction Patterns Across Configurations

The second scenario focuses on how predictions behave across different prediction configurations in the administrator view, in the *Prediction Comparison* section of Figure 2. Using the same graph and configuration, the demonstrator runs the prediction module several times, each time selecting a different prediction configuration. Each run produces a set of predicted edges on the same node set and an associated set of evaluation metrics.

Provlepsis4j summarizes the overlap between configurations in a heatmap view. For a fixed set of candidate edges, each cell reports the percentage of edges that are predicted by both configurations in the pair. High percentages indicate that the configurations make similar predictions, while low percentages mean that they select different edges. By inspecting these overlaps together with the reported metrics and re-running the same Cypher queries, this scenario shows how Provlepsis4j can be used to compare prediction configurations and see how their differences affect query answers on the predicted graphs.

Scenario 3: Querying Current and Predicted Graphs

The third scenario focuses on querying the current and predicted future graphs in the user view, shown in Figure 3. The demonstrator

Figure 3: Path Query example of Provlepsis4j (*Scenario 3*).

writes a Cypher query once and the system executes it on the current and predicted graph states, showing the results side by side as tables and graph visualizations. In the predicted graph panel, edges are color coded by their prediction timestamp, following the same green to red scale shown in the *Present-Future* bar at the top of the interface. When a shortest path or neighborhood is highlighted, users see which parts of the answer rely on edges from earlier prediction rounds and which rely on edges from later rounds.

This scenario demonstrates how the Provlepsis4j interface makes differences between the two graphs explicit. Using a shortest path query between two selected nodes, it shows whether the nodes are already connected in the current graph or become connected only after predicted edges are added, and how the path length changes. By interacting with the visualizations, users can inspect the nodes and relationships along each path in both graphs and relate any new or shorter paths in the predicted graph to the structure of the current graph.

4 RELATED WORK

Graph databases such as Neo4j typically evaluate queries only on the current graph. Recent work has improved query processing for labeled property graphs [3], but it does not address querying possible future graphs. Prior work on temporal graphs in databases has focused on present and historical data, including temporal aggregation and analytics [11] and temporal queries over timestamped and historical graphs, such as temporal paths, reachability, and time travel in graph databases [1, 8, 9, 12]. In our earlier work on future-time temporal path queries [4], we predicted shortest paths at future time points by combining temporal query processing with prediction oracles.

Link prediction methods estimate the probability of future edges from structure and attributes and are widely used in static and dynamic graphs [5, 7]. In many systems they are exposed as offline pipelines whose outputs are not directly available to the query engine. Neo4j GDS is a notable exception, since it provides node embedding and link prediction algorithms inside the database.

Provlepsis4j builds on the link prediction functionality of Neo4j GDS. It materializes predictions as persistent predicted graph states and allows the same Cypher query to be executed on the current and predicted states. The interface shows the two answers side by side, making explicit how predicted edges affect query results. This integration of prediction and querying is the focus of the demo.

5 CONCLUSIONS

In this demo, we presented *Provlepsis4j*, a system for querying both the current and a predicted future graph in Neo4j using the same Cypher queries. Provlepsis4j maintains current and predicted graph states and stores prediction timestamps on predicted edges. It provides an administrator view for model configuration and evaluation, and a user view for side-by-side querying. The administrator view reports standard evaluation metrics for each prediction configuration and shows prediction overlaps with heatmaps.

For future work we plan to support larger graphs and more complex query workloads, and extend the analysis views so that users can better understand the structure of the predicted graphs and their effect on query results.

ACKNOWLEDGMENTS

This work has been supported by the Hellenic Foundation for Research and Innovation (HFRI) under the 5th Call for HFRI PhD Fellowships (Fellowship Number: 20770) and by project MIS 5154714 of the National Recovery and Resilience Plan Greece 2.0 funded by the European Union under the NextGenerationEU Program.

REFERENCES

[1] Takuya Akiba, Yoichi Iwata, and Yuichi Yoshida. 2014. Dynamic and historical shortest-path distance queries on large evolving networks by pruned landmark labeling. In *23rd International World Wide Web Conference, WWW*. ACM.
[2] Haochen Chen, Syed Fahad Sultan, Yingtao Tian, Muhao Chen, and Steven Skiena. 2019. Fast and Accurate Network Embeddings via Very Sparse Random Projection. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*.
[3] James Clarkson, Georgios Theodorakis, and Jim Webber. 2024. BIFROST: A Future Graph Database Runtime. In *2024 IEEE 40th International Conference on Data Engineering (ICDE)*.
[4] Christos Gkartzios and Evaggelia Pitoura. 2023. Future-Time Temporal Path Queries *(GRADES-NDA '23)*.
[5] Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable Feature Learning for Networks. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*.
[6] William L. Hamilton, Rex Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*.
[7] Srijan Kumar, Xikun Zhang, and Jure Leskovec. 2019. Predicting Dynamic Embedding Trajectory in Temporal Interaction Networks. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD*.
[8] Konstantinos Semertzidis and Evaggelia Pitoura. 2016. Time Traveling in Graphs using a Graph Database. In *EDBT/ICDT Workshops*.
[9] Konstantinos Semertzidis and Evaggelia Pitoura. 2017. Historical Traversals in Native Graph Databases. In *Advances in Databases and Information Systems*. Springer International Publishing.
[10] Junhao Shen, Mohammad Ausaf Ali Haqqani, Beichen Hu, Cheng Huang, Xihao Xie, Tsengdar Lee, and Jia Zhang. 2024. Temporal graph neural network-powered paper recommendation on dynamic citation networks. *arXiv preprint arXiv:2408.15371* (2024).
[11] Evangelia Tsoukanara, Georgia Koloniari, and Evaggelia Pitoura. 2023. GraphTempo: An aggregation framework for evolving graphs. In *Proceedings 26th International Conference on Extending Database Technology, EDBT 2023, Ioannina, Greece, March 28-31, 2023*. OpenProceedings.org.
[12] Huanhuan Wu, Yuzhen Huang, James Cheng, Jinfeng Li, and Yiping Ke. 2016. Reachability and time-based path queries in temporal graphs. In *32nd IEEE International Conference on Data Engineering, ICDE*. IEEE Computer Society.
[13] Wei Wu, Bin Li, Chuan Luo, and Wolfgang Nejdl. 2021. Hashing-Accelerated Graph Neural Networks for Link Prediction. In *Proceedings of the Web Conference 2021*.