Future-Time Temporal Path Queries

Christos Gkartzios
Evaggelia Pitoura
Computer Science and Engineering Department, University of Ioannina
Greece

ABSTRACT

Most previous research considers processing queries on the current or previous states of a graph. In this paper, we propose processing future-time graph queries, i.e., predicting the output of a query on some future state of the graph. To process future-time queries, we present a generic approach that exploits a predictive model that provides oracles about the future state of the graph. We focus on future-time shortest path queries that given a temporal graph and two nodes return the shortest path between them at some future time. We present two algorithms each invoking a different type of oracle: (a) a link prediction oracle that given two nodes returns the probability of an edge between them, and (b) a connection prediction oracle that given a node u and a future time instance t returns the node v that u will connect to at t. Finally, we present experimental results using off-the-shelf prediction models that provide such oracles.

CCS CONCEPTS

• **Information systems** → **Data management systems**; • **Computing methodologies** → *Machine learning*.

KEYWORDS

temporal graphs, temporal queries, graph embeddings

ACM Reference Format:
Christos Gkartzios and Evaggelia Pitoura. 2023. Future-Time Temporal Path Queries. In *Joint Workshop on Graph Data Management Experiences & Systems (GRADES) and Network Data Analytics (NDA) (GRADES & NDA '23), June 18, 2023, Seattle, WA, USA.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3594778.3594879

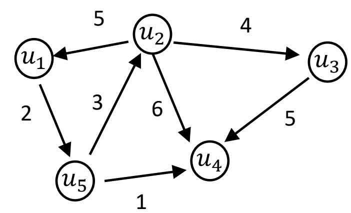

Figure 1: Shortest temporal paths: The shortest path between u_1 and u_4 in G_5 is $u_1u_5u_2u_3u_4$ and in G_6 $u_1u_5u_2u_4$.

1 INTRODUCTION

Graphs are ubiquitous data structures for modeling entities and the relationships between them. For example, in social networks, nodes correspond to people and edges to interactions, or relations between them. In transportation networks, nodes may represent cities and edges flights, or rides between them. Other examples include communication, cooperation, and biological networks. We are interested in temporal graphs, where each edge is annotated with information about the time the corresponding interaction, relationship, communication, transportation or cooperation appeared.

Most research on temporal graphs focuses on processing a graph query on the current state of the graph e.g., [18, 19], or on past states of an evolving graph e.g., [1, 6, 16]. Instead, we focus on predicting the output of a graph query at some future time point. We call such queries *future-time queries.* In particular, in this paper we consider future-time shortest temporal path queries that given a temporal graph and two nodes return the shortest path between them at some future time.

To process future-time graph queries, we propose an incremental query processing approach that leverages a prediction model M that provides oracles about the future state of the graph. Initially, we process the query up to the current time instance. Then, we invoke the oracle to extend processing at future time points. We consider prediction models that provide two types of oracles: (a) a link prediction oracle that given two nodes returns the probability of an edge between them, and (b) a connection prediction oracle that given a node u and a future time instance t returns the node v that u will connect to at t.

We have implemented our approach using off-the-shelf prediction models that provide such oracles. For the link prediction oracle, we use Node2vec [4] and Multilens [7] and for the connection prediction oracle, we use Jodie [9]. We present an initial experimental evaluation of the efficiency and quality of the approach using three real datasets.

The rest of this paper is structured as follows. In Section 2, we define the problem, while in Section 3, we present our approach. In Section 4, we provide experimental results, in Section 5 related work and in Section 6 conclusions and directions for future work.

2 FUTURE-TIME QUERIES

Temporal graphs are graphs where each edge is annotated with information regarding the time of occurrence of the interaction or relationship represented by the edge.

Definition 2.1. A temporal graph $G = (V, E)$ is a directed graph where V is the set of nodes and E is the set of edges of G. Each edge in E is a triple (u, v, t) with $u, v \in V$ and t is the time instance when the edge appeared.

GRADES & NDA '23, June 18, 2023, Seattle, WA, USA

ACM ISBN 979-8-4007-0201-3/23/06.
https://doi.org/10.1145/3594778.3594879

There may be multiple edges between two nodes with different timestamps. In the following, we will use the notation $G_T = (V, E_T)$ for the graph up to time T, that is for the graph that includes all edges (u, v, t) with $t \leq T$. We will use *now* to denote the current time instance.

Problem Definition. [Future-time graph queries] *Given temporal graph $G_{now} = (V, E_{now})$, time instance $\tau > now$, a prediction model M and a query q, predict the results of q on $G_\tau = (V, E_\tau)$.*

In this paper, we will focus on time-ordered paths [18].

Definition 2.2. A temporal path P of length k on graph $G_T = (V, E_T)$ is a path (u_1, u_2, … u_{k+1}), where $(u_i, u_{i+1}, t_i) \in E_T$ and $t_i < t_{i+1}$, for 1 ≤ i < k.

A temporal path P between nodes u and v is called *shortest* temporal path if there is no other temporal path between u and v with a length smaller than k.

Shortest temporal paths change with time as shown in Figure 1. The shortest temporal path between u_1 and u_4 is of length 4 at G_5, and of length 3 in G_6. We are interested in predicting the length of such temporal paths. Concretely, given G_{now}, a prediction model M, time instance $\tau > now$, and two nodes u and v, we want to predict the shortest temporal path between u and v in time instance τ. It is easy to show that the length of the shortest temporal path between two nodes in $G_{T'}$ is always smaller or equal to the length of their shortest path in G_T, for $T' > T$.

3 PROCESSING FUTURE-TIME QUERIES

To process future-time graph queries, we propose an incremental query processing approach that leverages a prediction model M. We explore two different prediction models, one providing a *link prediction* oracle and one providing a *connection prediction* oracle. Given G_{now} and a future time instance τ, (1) a link prediction oracle, $L(u, v)$, takes as input nodes u, v and returns the probability of an edge (u, v, t), $now < t \leq \tau$, while (2) a connection prediction oracle, $C(u, t)$, takes as input a node u and a time instance t, $now < t \leq \tau$, and returns the node v such that (u, v, t) in G_τ.

To predict the results of a future-time query q, the query is processed on G_{now} and then the oracle is invoked a number of times so as to predict the result of q in G_τ. Thus, the processing cost of a future-time query consists of the cost of building the model M, processing q on G_{now} and invoking the oracles.

We now describe the application of this general framework in the case of future-time shortest temporal path queries. For predicting the future-time shortest temporal paths from a node u to node v, the computational part on G_{now} consists of finding the shortest temporal paths from source node u to all nodes in G_{now} as shown in Algorithm 1.

Algorithm 1 takes as input a stream S of edges (u, v, t) ordered by time. It computes the shortest temporal paths from input node x to all other nodes $v \in V$ in a single pass of this stream. For each node v in the graph, the algorithm uses a list L_v, comprised of $(d[v], a[v], P[v])$ elements, where $P[v]$ is the shortest temporal path and $d[v]$ the shortest distance from x to v at time instance $a[v]$. L_v is sorted by distance. The algorithm takes advantage of the fact that if P is the shortest temporal path between two nodes u_1 and v_k in a time interval $[t_1, t_2]$, then every prefix subpath P_i of P to node v_i is the shortest path for v to v_i in the time interval $[t_1, end(P_i)]$, where $end(P_i)$ is the time instance of the last edge in P_i. In addition, in Line 19, we remove dominated elements from the L_v lists. An element $(d_1[v], a_1[v], P_1[v])$ is dominated by an element $(d_2[v], a_2[v], P_2[v])$ in L_u if $d_2[v] < d_1[v]$ and $a_2[v] \leq a_1[v]$, or $d_1[v] = d_2[v]$ and $a_2[v] < a_1[v]$.

Algorithm 1 Single source shortest temporal paths in G_{now}

1: **Input**: Source node x, edge stream S
2: **Output**: For each $v \in V$, $f[v] = (d[v], P[v])$ where $d[v]$ the shortest distance and $P[v]$ the shortest path from x to v
3: **Foreach** $v \in V$:
4: create empty L_v
5: Initialize $f[x] = (0, [x])$, and $f[v] = (\infty, [])$, for all $v \in V \backslash x$
6: **Foreach** incoming edge $e = (u, v, t)$ in S:
7: **If** $u == x$:
8: **If** $(0, t, []) \notin L_x$:
9: Insert $(0, t, [])$ into L_x
10: Let $(d'[u], a'[u], P'[u])$ be the element in L_u s.t.
11: $a'[u] = max(\{a[u] : (d[u], a[u], P[u]) \in L_u, a[u] \leq t\})$
12: $d[v] = d'[u] + 1$
13: $a[v] = t$
14: $P[v] = P'[u].append(v)$
15: **If** $a[v]$ in L_v:
16: Update the corresponding $d[v], P[v]$ in L_v
17: **else**:
18: Insert $(d[v], a[v], P[v])$ into L_v
19: Remove dominated elements in L_v
20: $d'[v] = f.d[v]$
21: **If** $d[v] < d'[v]$:
22: $f[v] = (d[v], P[v])$
23: return $f[v]$ for each $v \in V$

We now discuss how to use the oracles to predict future shortest temporal paths from source node u to destination node v. One issue with the link prediction oracle $L(u_1, u_2)$ is that it provides the probability of an edge from u_1 to u_2 appearing in some future time instance but not the exact time instance that this edge will appear. Thus, using this oracle, we are restricted to predicting only future temporal paths with a single future edge, since we cannot ensure the correct temporal order between future edges. Let d_{now} be the length of the shortest temporal path between u and v in G_{now}. Since the length of a shortest temporal path in G_τ is smaller or equal to d_{now}, we invoke the oracle $L(w, v)$ for all nodes w at a distance at most d_{now} form u in G_{now}. We return the path with the highest probability. These steps are shown in Algorithm 2.

The connection prediction oracle $C(u_1, t)$ returns the node u_2 that u_1 will connect to at time instance t. Although the connection prediction oracle returns the exact time of the edge, to be comparable with the link prediction oracle, we again restrict our task to predicting future-time temporal shortest paths with a single future edge. We invoke $C(w, t)$ for all time instances t in $(now, \tau]$ and all nodes w with distance at most d_{now} form u in G_{now}. Each time, we check whether the predicted node is the destination node v. To do so, we assume a similarity function *sim* between nodes. In the specific prediction model that we use in our implementation the

Algorithm 2 Future-time shortest temporal path prediction using the link prediction oracle

Input: Source node u, destination node v, $f(w) = (d(w), P(w))$ for each w in V and d_{now} as computed by Alg. 1 with u as source node, future time instance τ
Output: Predicted shortest temporal path P from u to v in G_τ
ForEach: node w with $f.d(w) \leq d_{now}$:
    $p(w, v) = L(w, v)$
Let w be the node with the largest $p(w, v)$
$P = P(w) \cdot (w, v)$
Return P

similarity function corresponds to cosine similarity between the embedding of the predicted node $C(w, t)$ and the embedding of v. We select the edge whose returned node is the most similar to v. We also return the corresponding time instance. These steps are shown in Algorithm 3.

Algorithm 3 Future-time shortest temporal path prediction using the connection prediction oracle

Input: Source node u, destination node v, $f(w) = (d(w), P(w))$ for each w in V and d_{now} computed by Alg. 1 with u as source node, future time instance τ
Output: Predicted shortest temporal path P from u to v in G_τ, predicted time instance t'
ForEach time instance $t \in (now, \tau]$:
    ForEach: node w with $d(w) \leq d_{now}$:
        $x = C(w, t)$
        $s(w, t) = sim(x, v)$
    Let (w, t') be the prediction with the smallest $s(w, t)$ over all t
Let (w', t') be the prediction with the smallest $s(w, t')$ over all nodes w
$P = P(w') \cdot (w', v)$
Return P
Return t'

4 EXPERIMENTAL EVALUATION

We present an initial evaluation of our approach using the following real datasets whose characteristics are summarized in Table 1:

CollegeMsg[1]: a network of messages sent in an online social network. Nodes represent participants in a conversation and edges denote messages sent at a specific timestamp. Timestamps are in seconds and correspond to 193 days.

Enron[2]: a network based on the exchange of emails between the employees of Enron. Nodes represent the employees and edges the emails sent. Timestamps are in seconds and correspond to a time interval from May, 1999 to June, 2002.

Bitcoin[3]: a network of the interactions between people who trade using Bitcoin. Nodes correspond to the humans trading with bitcoin and edges correspond to the trust rating interactions between users. Timestamps are in seconds.

[1]https://snap.stanford.edu/data/CollegeMsg.html
[2]https://networkrepository.com/ia-enron-employees.php
[3]https://snap.stanford.edu/data/soc-sign-bitcoin-alpha.html

Table 1: Dataset characteristics

	CollegeMsg	Enron	Bitcoin
Nodes	1,899	150	3,783
Static Edges	13,838	1,526	14,124
Temporal Edges	59,835	47,088	24,186
Timestamps	58,911	14,832	1,647

For the prediction model that provides a link prediction oracle, we utilize embedding-based methods. Embeddings are produced using two methods, a non-temporal embedding, namely Node2vec [4] and a temporal one, namely Multilens [7]. We use the node embeddings as input to a link prediction classifier.

For the connection prediction oracle, we use a model generated by Jodie [9]. Jodie exploits the interactions between users and items/products. It builds separate embeddings for users and items. These embeddings are then used to predict future user-item interactions. To leverage Jodie in our setting, we build two embeddings for each node: an embedding of a node as a source node (corresponding to user embeddings) and an embedding of a node as a target node (corresponding to item embeddings).

Since Jodie works only for predicting positive (actual) interactions, we test our approach only on positive examples, i.e, on paths whose temporal shortest path distance did change in G_τ. We further restrict our tests to future paths with a single future edge, i.e., paths that include only nodes that exist in the temporal shortest paths in G_{now}. Thus in Algorithms 2 and 3 we invoke the oracles only on these nodes. Table 2 presents statistics about our test queries.

Table 2: Query characteristics

	CollegeMsg	Enron	Bitcoin
Number of queries	62	89	30
Average distance in G_{now}	2.79	2.56	2.46
Average distance in G_τ	1	1.1	1

To train the models, we partition the datasets by time. We train the models on the first 80% of the interactions. Specifically, for Node2vec and Multilens, we build embeddings using 80% of the interactions. We use these embeddings to build a link prediction classifier. As train set for this classifier, we use as positive (negative) examples the edges that appear (resp. do not appear) in the next 10% of the interactions. We use this classifier as our link prediction oracle. Note that we use again Node2vec and Multilens to produce embeddings for the 90% of the interactions and use these embeddings as input to the classifier for predicting future interactions. For Jodie we use 80% of the edges for training, the next 10% for validation and the last 10% for testing. That is in all cases, G_{now} includes the first 90% of the interactions.

We first present performance results in Table 3. We report separately: (1) the time to build the model (prepossessing cost), (2) the time to compute shortest temporal paths in G_{now} (average time per query) and (3) the time to predict the shortest paths in G_τ by invoking the corresponding oracles (average time per query). Specifically for the Node2vec and Multilens methods, we report the time needed for the computation of the embeddings at 80% of interactions, the

Table 3: Performance results (in secs)

Time to build the model									
	CollegeMsg			Enron			Bitcoin		
	embed. 80%	classifier	embed. 90%	embed. 80%	classifier	embed. 90%	embed. 80%	classifier	embed. 90%
Node2vec	35.57	0.39	41.66	2.15	0.25	2.45	57.74	0.1	61.76
Multilens	377.3	0.98	427	34.38	0.71	35	569.98	0.55	575.52
Jodie	35,749			18,114			33,399		
Processing time on G_{now} (Alg. 1)									
	0.035			0.007			0.162		
Time for future path prediction by invoking the prediction oracles (Alg. 2 and 3)									
Node2vec	1.53			1.17			1.17		
Multilens	1.31			1.21			1.27		
Jodie	1.19			1.19			1.04		

training of the classifier and the computation of the embeddings at 90% of the interactions. Multilens takes more time than Node2vec since it takes into account temporal information and Jodie is slower since it updates the representations interactively. Prediction times are comparable for this set of queries.

Table 4: Quality of future-time query predictions

Correct paths predicted			
	CollegeMsg	Enron	Bitcoin
Node2vec	52%	80%	90%
Multilens	71%	76%	80%
Jodie	87%	81%	97%
MSE of predicted distance			
Node2vec	0.64	0.16	0.2
Multilens	0.57	0.18	0.25
Jodie	0.42	0.22	0.04
Normalized MSE of predicted timestamps			
Jodie	0.32	0.15	0.16

We report results about the quality of the predictions in Table 4. Jodie outperforms the other methods in terms of predicting the actual paths. It also performs better in terms of the MSE of the distance of the predicted path. The only exception is the Enron dataset where Jodie misses some paths whose actual distance is short. Multilens outperforms Node2vec since it is able to explore temporal information in building the embeddings. We also report the normalized MSE of the timestamps predicted by Jodie. Specifically, we compute the MSE (MSE_{random}) of a random prediction that predicts that the timestamp of the edge will appear in the middle of the interval $[now, \tau]$. We report the ratio of the MSE of the timestamps predicted by Jodie and MSE_{random}. Jodie is quite successful in predicting the actual timestamps.

5 RELATED WORK

There is a lot of previous work on path queries on temporal graphs. Most previous research focuses on processing a path query on the current state of the graph e.g., [2, 18, 19], or on past states of an evolving graph e.g., [1, 6, 15–17]. In this paper, we focus on predicting the output of the query at some future time point. Our overall approach is based on combining query processing with invoking generic oracles. Alternatively, we can build an end-to-end prediction model pertinent to temporal path queries. This approach has the shortcoming that we need to build a different model for each type of query. Instead, our proposal advocates a general framework of extending graph query processing with ML oracles. There is some previous work for end-to-end models for estimating distances [5, 14]. Note that these works do not consider predicting future distances but instead estimating them in the current graph.

Our work falls in the general area of combining query processing and ML inference. There are several lines of research in this area including using database techniques to optimize ML pipelines, ML to improve database internals and speeding up ML invocations following either inside the database or application-site solutions e.g., [10–12]. A specific line of research considers approximate queries over ML models, i.e., by invoking cheap proxies e.g., [3, 8]. We propose a general framework for temporal predictions that can be part of a query processing pipeline opening new interesting problems of optimizing such pipelines. Finally, there is some relation with probabilistic graphs e.g., [13] where each edge is associated with a probability to appear as is the case for future edges.

6 CONCLUSIONS AND FUTURE WORK

In this paper, we propose processing future-time graph queries, i.e., predicting the output of a query on some future state of the graph. To process future-time queries, we advocate a generic approach that exploits a predictive model that provides oracles about the future state of the graph. In this paper, we have focused on shortest temporal path queries. There are various directions for future work. One direction is extending our predictions to future-time temporal paths with many future edges. Other directions include studying other types of graph queries, e.g., future-time graph pattern queries and other types of oracles. Other interesting issues include updating the prediction model and providing estimates about the accuracy of the prediction given the accuracy of the oracles.

ACKNOWLEDGMENTS

Research work supported by the Hellenic Foundation for Research and Innovation (H.F.R.I.) under the "1st Call for H.F.R.I. Research Projects to Support Faculty Members & Researchers and Procure High-Value Research Equipment" (Project Number: HFRI-FM17-1873, GraphTempo).

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, 237–248.

[2] Marcelo Arenas, Pedro Bahamondes, Amir Aghasadeghi, and Julia Stoyanovich. 2022. Temporal Regular Path Queries. In *38th IEEE International Conference on Data Engineering, ICDE 2022, Kuala Lumpur, Malaysia, May 9-12, 2022*. IEEE, 2412–2425.

[3] Dujian Ding, Sihem Amer-Yahia, and Laks V. S. Lakshmanan. 2022. On Efficient Approximate Queries over Machine Learning Models. *Proc. VLDB Endow.* 16, 4 (2022), 918–931.

[4] 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, San Francisco, CA, USA, August 13-17, 2016*. ACM, 855–864.

[5] Shuai Huang, Yong Wang, Tianyu Zhao, and Guoliang Li. 2021. A Learning-based Method for Computing Shortest Path Distances on Road Networks. In *37th IEEE International Conference on Data Engineering, ICDE*. IEEE, 360–371.

[6] Wenyu Huo and Vassilis J. Tsotras. 2014. Efficient temporal shortest path queries on evolving social graphs. In *Conference on Scientific and Statistical Database Management, SSDBM*. ACM, 38:1–38:4.

[7] Di Jin, Sungchul Kim, Ryan A. Rossi, and Danai Koutra. 2022. On Generalizing Static Node Embedding to Dynamic Settings. In *WSDM '22: The Fifteenth ACM International Conference on Web Search and Data Mining, Virtual Event / Tempe, AZ, USA, February 21 - 25, 2022*. ACM, 410–420.

[8] Daniel Kang, Edward Gan, Peter Bailis, Tatsunori Hashimoto, and Matei Zaharia. 2020. Approximate Selection with Guarantees using Proxies. *Proc. VLDB Endow.* 13, 11 (2020), 1990–2003.

[9] 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 2019*. ACM, 1269–1278.

[10] Guoliang Li and Xuanhe Zhou. 2022. Machine Learning for Data Management: A System View. In *38th IEEE International Conference on Data Engineering, ICDE*. IEEE, 3198–3201.

[11] Guoliang Li, Xuanhe Zhou, and Lei Cao. 2021. Machine Learning for Databases. *Proc. VLDB Endow.* 14, 12 (2021), 3190–3193.

[12] Kwanghyun Park, Karla Saur, Dalitso Banda, Rathijit Sen, Matteo Interlandi, and Konstantinos Karanasos. 2022. End-to-end Optimization of Machine Learning Prediction Queries. In *SIGMOD Conference*. ACM, 587–601.

[13] Michalis Potamias, Francesco Bonchi, Aristides Gionis, and George Kollios. 2010. k-Nearest Neighbors in Uncertain Graphs. *Proc. VLDB Endow.* 3, 1 (2010), 997–1008.

[14] Jianzhong Qi, Wei Wang, Rui Zhang, and Zhuowei Zhao. 2020. A Learning Based Approach to Predict Shortest-Path Distances. In *Proceedings of the 23rd International Conference on Extending Database Technology, EDBT*. OpenProceedings.org, 367–370.

[15] Konstantinos Semertzidis and Evaggelia Pitoura. 2016. Durable graph pattern queries on historical graphs. In *32nd IEEE International Conference on Data Engineering, ICDE 2016, Helsinki, Finland, May 16-20, 2016*. IEEE Computer Society, 541–552.

[16] Konstantinos Semertzidis, Evaggelia Pitoura, and Kostas Lillis. 2015. TimeReach: Historical Reachability Queries on Evolving Graphs. In *EDBT*. OpenProceedings.org, 121–132.

[17] 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, 221–233.

[18] Huanhuan Wu, James Cheng, Silu Huang, Yiping Ke, Yi Lu, and Yanyan Xu. 2014. Path Problems in Temporal Graphs. *Proc. VLDB Endow.* 7, 9 (2014), 721–732.

[19] 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, 145–156.