Towards a Temporal Graph Query Language for Durable Patterns

Daniel Betsche
daniel.betsche@kit.edu
Karlsruhe Institute of Technology (KIT)
Karlsruhe, Germany

Balduin Katzer
balduin.katzer@kit.edu
Karlsruhe Institute of Technology (KIT)
Karlsruhe, Germany
Karlsruhe University of Applied Sciences (HKA)
Karlsruhe, Germany

Katrin Schulz
katrin.schulz@kit.edu
Karlsruhe Institute of Technology (KIT)
Karlsruhe, Germany
Karlsruhe University of Applied Sciences (HKA)
Karlsruhe, Germany

Klemens Böhm
klemens.boehm@kit.edu
Karlsruhe Institute of Technology (KIT)
Karlsruhe, Germany

ABSTRACT

Dynamic graphs are often the initial data for scientific analyses. However, existing methods designed for static graphs struggle with efficiency and accuracy when applied dynamically. One challenge occurs when local interactions in dynamic graphs influence global phenomena. Practitioners then follow the evolution of relationships between individual elements in local structures. Such structures are called Durable Graph Patterns or evolving subgraphs. This work introduces the Durable Graph Pattern Query Language (DPQGL), which allows for user-friendly querying of durable graph patterns on dynamic graphs. DPGQL is, by design, agnostic to the underlying durable pattern-matching algorithm. We base our proposed language on the widely used Cypher Query Language. In our experiments with seven pattern shapes in 24 variations on real-world materials science data, we explore the impact on query runtimes from query complexity and the frequency of graph changes.

CCS CONCEPTS

• **Information systems → Query languages for non-relational engines**.

KEYWORDS

Temporal graph, Neo4j, Cypher query language, graph databases, query languages, durable graph patterns, graph query processing

ACM Reference Format:
Daniel Betsche, Balduin Katzer, Katrin Schulz, and Klemens Böhm. 2024. Towards a Temporal Graph Query Language for Durable Patterns. In *36th International Conference on Scientific and Statistical Database Management (SSDBM 2024), July 10–12, 2024, Rennes, France.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3676288.3676303

SSDBM 2024, July 10–12, 2024, Rennes, France

ACM ISBN 979-8-4007-1020-9/24/07
https://doi.org/10.1145/3676288.3676303

1 INTRODUCTION

In dynamic networks, large-scale behavior often emerges from the collective, intermittent interactions of the system's intrinsic small-scale elements. These elements can form lasting relationships in their local area, so-called *durable patterns* [16, 17], that impact the whole system. This article considers querying such patterns in dynamic graphs with frequently occurring changes, which is of great importance in scientific simulations and experiments.

Example 1.1 (Dislocation dynamics). Dislocations are line defects on the atomic length scale in a material's crystallographic structure. The collective motion of dislocations in solid materials results in plastic deformation [1]. To date, the plastic deformation behavior is not well understood for complex dislocation networks due to a large variety of dislocation interactions [9]. These interactions lead to the formation of *junctions*, which collectively change the material's properties while forming new dislocations, dissolving existing ones, or stabilizing the network. Understanding junctions' reaction properties within complex networks is key for materials scientists to predict where and when a material irreversibly deforms [3, 13].

To study such phenomena, scientists run simulations that calculate the behavior of the individual elements on a microscopic scale [5, 19]. Such simulations result in temporal graph data $\mathcal{G}_{[i,j]}$, i.e., sequences $G_i, G_{i+1}, \ldots G_j$ of static graphs over changing sets of nodes $\mathcal{N}$, edges $\mathcal{E}$, and properties $\mathcal{K}$. Connecting back to the example, each microscopic element is modeled as a node with a list of properties, an identifier, and a type label. A straightforward approach to identifying durable patterns is to apply a pattern-matching algorithm to every graph snapshot $G_t \in \mathcal{G}_{[i,j]}$ and to aggregate the results into series of subgraphs.

A stable junction [12, 15] is a special type of dislocation, acting as a barrier against other moving dislocations. Listing 1 and 2 show a Cypher and a DGPQL query for mobile dislocations attached to stable junctions, whose length and curvature can change over time. Their evolution is of particular interest as they play a key role in understanding how and when these barriers can dissolve again.

After querying the graph as in Listing 1, the user must still aggregate the patterns from individual snapshots into durable patterns. In order to do so, three points remain to be done: (1) identifiers of all nodes and edges in the pattern are required across all its states,

(2) redundant patterns must be identified and pruned, requiring constraints on the identifiers or isomorphism checks for the queried subgraphs, and (3) feeding the patterns to an algorithm identifying the durable patterns.

```
MATCH (p1)-[l1:Link]-(p2)-[j:Junction]-(p3)-[l2:
    Link]-(p4)
WHERE p1.id <> p2.id <> p3.id <> p4.id
and l1.id <> l2.id and l1.id < l2.id
and j.type = 1
and n.time >= 500 and n.time <= 2000
RETURN p1.id, p2.id, p3.id, p4.id, j.id,
       l1.id, l1.time, l1.length, l1.curvature,
       l2.id, l2.time, l2.length, l2.curvature
```

Listing 1: Cypher query for mobile dislocations $l1$ and $l2$.

```
MATCH ()-[l1:Link]-()-[j:Junction]-()-[l2:Link]-()
    [500, 2000]
WHERE j.type = 1
RETURN l1(time, id, length, curvature),
       l2(id, length, curvature)
```

Listing 2: DPQGL equivalent of the Cypher query.

Our proposed query language performs (1), (2) and (3) transparently for the user. In addition, it is agnostic of the underlying data model by having time intervals as part of the query pattern (see Listing 2, Line 1). Therefore, a query engine can choose the most promising durable pattern-matching algorithm as part of physical query optimization. Specifically, our contributions are as follows:

We propose and formalize the Durable Graph Pattern Query Language (DGPQL), a query language that facilitates concise and flexible queries for durable patterns in dynamic graphs.

We experimentally show runtimes of querying durable graph patterns using real-world data. We queried 20 durable patterns from seven common pattern shapes [4] to explore how pattern complexity and lifespan impact query runtimes.

2 INTRODUCING DGPQL

We first introduce DGPQL by example. Figure 1 shows a dynamic Graph $\mathcal{G}_{[1,4]}$ representing a simplified excerpt of dislocation movement data. For simplicity, we include only one node and two edge types, junctions `:J`, and links `:L`. A junction is an edge where two dislocations interact. A link is a freely moving dislocation connected to junctions at both ends.

```
MATCH (p1)-[j1:J]->(p2)-[j2:J]->(p3)[2,3]
RETURN j1(id, ...),j2(id, ...)
```

Listing 3: Querying a double junction between t_2 and t_3.

To illustrate the temporal aspect of DGPQL, we query for double junctions, i.e., places where multiple dislocations meet each other, in Listing 3. A query over window `[1,4]` would return an empty result, as no such durable pattern exists. For the window `[2,3]` the result contains the double junctions $(j_{1,4}, j_{4,7})$, $(j_{2,1}, j_{1,3})$, and $(j_{5,2}, j_{2,1})$. Increasing our window to either `[1,3]` or `[2,4]` will return a subset of these patterns. Note that, while we are only interested in the properties of the junctions, the nodes `p1,p2,p3` are still relevant to identify the durable patterns. Users can specify fixed windows

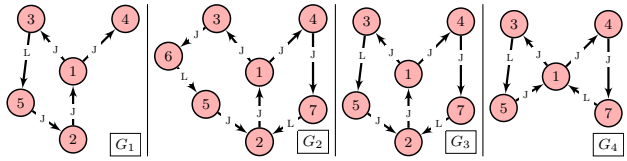

Figure 1: Temporal graph $\mathcal{G}_{[1,4]}$ with a small set of changing nodes and edges.

`[t1,t2]`, single time instants `[t]`, open windows with `[t,]` and `[,t]`, or omit the statement entirely to query over the graphs full duration.

```
MATCH p = (p1)--(p2)--(p3),(p2)--(p4)[t,t+1]
MATCH (p2)-[*2..4]-(p5) [t-1,t]
OPTIONAL MATCH (p2)--(p6)
RETURN p, p5, p6
```

Listing 4: Querying a pattern with variable length and optional components.

Listing 4 shows how to create complex query patterns by connecting patterns with at least one shared variable, here `p2`. We can do so by specifying multiple `MATCH` statements (Line 1-2) or using the same comma operator (Line 1) as Cypher does to chain pattern strings together. If several `MATCH` statements specify different query windows, only the overlap of both windows is evaluated. The query will yield the Cartesian product of both patterns if no variable representing a node or an edge occurs in the distinct patterns. To allow variable length graph structures, we use wildcards for relationships and optional components for patterns (Listing 4, Line 2-3).

3 FORMAL SPECIFICATION AND QUERY EVALUATION

Our design bases on Cypher [7, 8], the high-level query language of Neo4j. We limit our discussion to our modifications; see [8] for the full specification. To enable temporal graph processing and durable pattern matching our data model extends Cypher's model with temporal graphs and replaces its tables with durable patterns. Both languages consist of *expressions*, *patterns*, *clauses*, and *queries*. The syntax for expressions and patterns remains unchanged, we modify the clauses and queries by removing incompatible language aspects and adding support for temporal specifications.

Our core idea is to allow, given a temporal property graph $\mathcal{G}$, a pattern π, and a query window τ under an assignment u of values, to find all durable patterns $m \in M$ that satisfy the matching relation. We write the pattern matching relation as $(p, \mathcal{G}, u, \tau) \models \pi$. To find an assignment u of values for an expression $expr$ on a graph $\mathcal{G}$, we define the semantics of expressions as $[\![expr]\!]_{\mathcal{G},u}$. Lastly, we have the semantics of clauses and queries, where a query Q (or clause C) on a graph $\mathcal{G}$ is associated with a function $[\![Q]\!]_{\mathcal{G}}$ which takes a set of durable patterns $\mathcal{M}$ and returns a modified set of patterns.

3.1 Data Model

An interval τ specifies query windows or represents the lifetimes of nodes and edges. A temporal graph is associated with a set $\mathcal{I}$ of intervals for every node and edge. We ask that a set of time intervals

$$
\begin{aligned}
query &::= \text{RETURN } ret \mid clause\ query \\
ret &::= * \mid ret\_node \mid ret,\ ret\_node \\
ret\_node &::= a \mid a(ret\_prop) \qquad (1) \\
ret\_prop &::= k \mid ret\_prop,\ k \\
clause &::= [\text{OPTIONAL}]\text{MATCH } p\_tuple\ [\text{WHERE } expr] \qquad (2) \\
p\_tuple &::= pattern \mid pattern\ time \mid pattern,\ p\_tuple \\
time &::= [d] \mid [d\_1,] \mid [,d\_2] \mid [d\_1, d\_2]
\end{aligned}
$$

Figure 2: Syntax of queries and clauses.

$\mathcal{I}$ for a pattern m is overlapping and continuous [16, 17]. We use the shorthand $[i, j]$ for intervals $[t_i, t_j]$. We base our definition of temporal graphs on labeled property graphs (LPG; [8]).

Definition 3.1 (Labeled Temporal Property Graph; LTPG). An LTPG $\mathcal{G}_{[i,j]}$ in time interval $\tau = [i, j]$ is a sequence $\{G_i, G_{i+1}, \ldots, G_j\}$ of LPGs, also referred to as graph snapshots.

The lifespan of a node u or edge e is the duration it exists in an LTPG; it is a continuous interval with finite start and end.

Definition 3.2 (Durable Graph Pattern). A durable graph pattern m is an LTPG with the lifespan $\tau_m = \bigcap_{\tau_i \in \mathcal{I}_m} \{\tau_i\}$, i.e., the overlap of all lifespans for nodes and edges in $\mathcal{I}_m$.

A graph pattern query asks for all occurrences, or matches, of a user-specified graph pattern $P = (N_P, E_P, src_P, tgt_P, \iota_P, \lambda_P, \delta_P)$ in a graph $G = (N, E, src, tgt, \iota, \lambda, \delta)$; see [8] for details.

Definition 3.3 (Durable Graph Pattern Matching). Given an evolving graph $\mathcal{G}_{[i,j]}$, a graph pattern query P, a set of of time intervals $\mathcal{I}_p$, and a query window τ, a continuous-time durable graph pattern query find the subgraphs m of $\mathcal{G}$ such that (1) there exists a bijective mapping $f : N_p \to N_m$ such that $\forall n \in N_p, \lambda_p(v) \subseteq \lambda_m(f(v))$ and for each edge $e \in \mathcal{E}_p$, $f(e) \in \mathcal{E}_m$, and (2) $lspan(m, P, \mathcal{G}_{[i,j]}) \otimes \mathcal{I}_p \subseteq \tau$ is continuous and exists during the full query window.

3.2 Syntax and Semantics

Figure 2 shows the syntax of clauses and queries. A query is a sequence of clauses that ends with a **RETURN** statement. In our language, the return is a list of names for nodes and edges, see (1). A second list can follow these names, further specifying which property keys to return. The semantics are collected in Figure 3, with queries Q and clauses C. The set of durable patterns $\mathcal{M}$ contains all matched patterns.

DPGQL clauses are functions that, relative to an LTPG $\mathcal{G}$, take a set of durable graph patterns $\mathcal{M}$ and return the modified set based on the function and its parameters. Clauses only require a **MATCH** statement, see (2), which returns the values as sequences for the matched durable patterns. The **OPTIONAL** and **WHERE** statements can be omitted.

4 EXPERIMENTS

Our dataset from materials science is a simulation of dislocation dynamics. The simulation mimics an irreversible deformation of an

Queries:

$$
\begin{aligned}
&[\![RETURN *]\!]_{\mathcal{G}}(\mathcal{M}) = \mathcal{M} \text{ if has at least one entry} \\
&[\![RETURN\, e_1, \ldots, e_m]\!]_{\mathcal{G}}(\mathcal{M}) = \{(e_1, \ldots, e_m) \mid m \in \mathcal{M} \\
&\qquad\qquad \wedge (e_1 \ldots e_m) \subseteq m\} \\
&[\![C\ Q]\!]_{\mathcal{G}}(\mathcal{M}) = [\![Q]\!]_{\mathcal{G}}\left([\![C]\!]_{\mathcal{G}}(\mathcal{M})\right) \\
&[\![MATCH \pi]\!]_G(\mathcal{M}) = \bigcup_{u \in \mathcal{M}} \{m \in match(\pi, \mathcal{G}, u, \tau)\}
\end{aligned}
$$

Clauses:

$$
\begin{aligned}
&[\![MATCH\ \pi\ WHERE\ expr]\!]_G(\mathcal{M}) \\
&\qquad = [\![WHERE\ expr]\!]\Big([\![MATCH\ \pi]\!]_G\ \mathcal{M})\Big) \\
&[\![OPTIONAL\ MATCH\ \pi\ WHERE\ expr]\!]_G(\mathcal{M}) \\
&\quad = \bigcup_{u \in \mathcal{M}} \begin{cases} [\![MATCH\ \pi\ WHERE\ true]\!]_G(\{u\}) & \text{if} \neq \emptyset \\ (u, (free(u, \pi) : null)) & \text{otherwise} \end{cases} \\
&[\![OPTIONAL\ MATCH\ \pi]\!]_G(\mathcal{M}) \\
&\qquad = [\![OPTIONAL\ MATCH\ \pi\ WHERE\ true]\!]_G(\mathcal{M}) \\
&[\![WHERE\ expr]\!]_G(\mathcal{M}) = \{u \in \mathcal{M} \mid [\![expr]\!]_{G,u} = true\}
\end{aligned}
$$

Figure 3: Semantics of queries and clauses.

Table 1: Seven pattern shapes in 20 variants.

Name	Variants	#
Single Edge	-	1
Chain	3 to 6 nodes	4
Tree	One or two branches	2
Star	3 to 5 satellite nodes	3
Cycle	3 to 5 nodes	3
Flower	at least 2 of petal, stem, and stamen	4
Clique	3 to 5 nodes	3

aluminum cube of 2 µm length. The initial configuration consists of 400 randomly seeded dislocations, resulting in a high volume of interactions over 95 temporal snapshots. The dislocations are then transferred into a property graph G_t [11], which consists of nodes with physical properties connected by links or junctions.

We created a comprehensive list of the most common graph patterns, with Table 1 giving an overview of all queried patterns. Our list extends [4] with cliques, which also frequently occur in our context [14, 16, 17]. Single edges have no variants. Chains are three or more nodes in a row. A tree is a pattern where every node has exactly one path to any other node. A single-branch variant is a tree with five nodes over three levels, and two branches are seven nodes over three levels. A star has a center node to which every satellite node connects. A cycle is a chain where the last node connects to the first node. [4] defines a flower as a node with at least two of three attachments; *petals* (cycles), *stamens* (chains), or *stems* (trees that are not chains). The variations of the flower pattern consist of either tree and chain (flower12), tree and petal (flower13), chain and petal (flower23), or all three attachments (flower123).

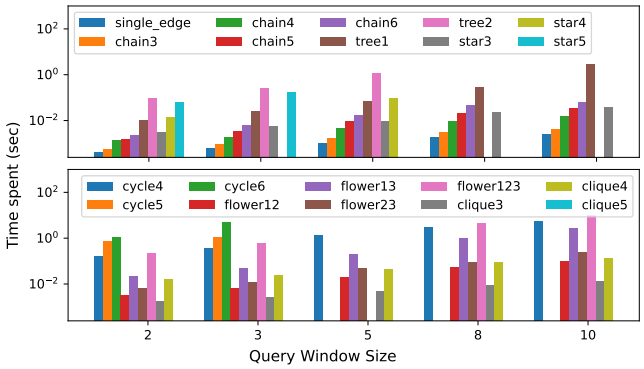

Figure 4: Average query runtimes for all pattern variants.

We ran our experiments with the baseline [17] and the pattern-matching of Neo4j 5.15. We evaluate the runtime needed to find durable patterns for all 20 variations of our seven shapes over five query windows. To obtain the average, we sample five query windows per pattern. We limited the matched patterns to 10,000 per snapshot to reduce experiment runtime. We further limited our query windows to no more than a lifespan of 10, as too few patterns existed longer.

Our results are in Figure 4. One sees that more complex queries tend to require more time and match our expectations. While the most complex pattern—flower123—takes the longest, it is surprising that flower13 and cycle4 are relatively close. Cycle5 and cycle6 are only found in query windows with size ≤ 3 but require significantly longer computation than any other patterns. We hypothesize that the cause for these runtimes is due to inefficient processing of their permutations. On the other hand, one observes striking differences in runtime between patterns with the same amount of nodes and edges but otherwise different configurations, e.g., tree1, tree2, and flower variants with trees. We suppose that the increased number of possible paths between nodes is the reason for these increases.

5 RELATED WORK

A baseline and an improved algorithm for durable graph pattern-matching were introduced for mining stable author groups in citation networks [16, 17]. The former finds matches in each static graph snapshot and then orders each match's identifiers, reducing the problem to string matching. The latter transforms the snapshot graph into a labeled version graph and uses several indexes to filter and refine a candidate set to identify durable patterns. A recent tree-based durable subgraph matching algorithm combined with a query decomposition method [14] improves upon their performances.

Modern graph databases and their query languages such as Cypher [8], SPARQL [18], or G-CORE [2] are aimed at static graphs and only offer limited support for temporal graphs. Graph databases to store temporal graphs exist [10], but focus on efficient storage and quick retrieval of snapshots over arbitrary time windows for static graph queries. To our knowledge, the only other temporal graph query language is T-GQL [6]. It is designed to solve the specific problem of temporal path queries, e.g., the fastest-, earliest-, shortest-, or latest-departure path; it does not address our use case.

6 CONCLUSIONS

Current limitations regarding mining dynamic graphs are twofold: First, existing query languages lack support for temporal queries. Second, graph database systems can not handle temporal graphs, let alone in a unified fashion. In this work, we tackle the former and propose the high-level query language DPGQL for temporal graphs, which facilitates the mining of durable graph patterns. DPGQL is, by design, agnostic of the underlying durable pattern-matching algorithm. Hence, advances in these algorithms directly benefit existing queries.

ACKNOWLEDGMENTS

The authors thank Daniel Weygand for the discussion and support in generating the dislocation data. This work was supported by the DFG (project no. 452183896), and by the Ministry of Science, Research and the Arts Baden-Württemberg project: "Algorithm Engineering for the Scalability Challenge (AESC)".

REFERENCES

[1] G. Ananthakrishna. 2007. Current theoretical approaches to collective behavior of dislocations. *PHYS REP* 440, 4–6 (mar 2007), 113–259.
[2] R. Angles, M. Arenas, P. Barcelo, P. Boncz, G. Fletcher, C. Gutierrez, T. Lindaaker, M. Paradies, S. Plantikow, J. Sequeda, O. van Rest, and H. Voigt. 2018. G-CORE: A Core for Future Graph Query Languages. In *Proc. of SIGMOD*. 1421–1432.
[3] N. Bertin, L.A. Zepeda-Ruiz, and V.V. Bulatov. 2022. Sweep-tracing algorithm: in silico slip crystallography and tension-compression asymmetry in BCC metals. *Materials Theory* 6, 1 (jan 2022).
[4] Angela Bonifati, Wim Martens, and Thomas Timm. 2017. An analytical study of large SPARQL query logs. *Proceedings of the VLDB Endowment* 11, 2 (Oct. 2017), 149–161. https://doi.org/10.14778/3149193.3149196
[5] V. Bulatov and W. Cai. 2013. *Computer Simulations of Dislocations*. Oxford University Press, Oxford, New York.
[6] A. Debrouvier, E. Parodi, M. Perazzo, V. Soliani, and A. Vaisman. 2021. A Model and Query Language for Temporal Graph Databases. *The VLDB Journal* 30, 5 (Sept. 2021), 825–858.
[7] N. Francis, A. Green, P. Guagliardo, L. Libkin, T. Lindaaker, V. Marsault, S. Plantikow, M. Rydberg, M. Schuster, P. Selmer, and A. Taylor. 2018. Formal Semantics of the Language Cypher. arXiv:1802.09984 [cs]
[8] N. Francis, A Green, P. Guagliardo, L. Libkin, T. Lindaaker, V. Marsault, S. Plantikow, M. Rydberg, P. Selmer, and A. Taylor. 2018. Cypher: An Evolving Query Language for Property Graphs. In *Proc. of SIGMOD*. 1433–1445.
[9] J. P. Hirth. 1961. On Dislocation Interactions in the fcc Lattice. *J APPL PHYS* 32, 4 (apr 1961), 700–706.
[10] A.P. Iyer, Q. Pu, K. Patel, J.E. Gonzalez, and I. Stoica. 2021. TEGRA: Efficient Ad-Hoc Analytics on Evolving Graphs.. In *NSDI*. 337–355.
[11] B. Katzer, D. Betsche, K. Böhm, D. Weygand, and K. Schulz. 2024. A Graph Database for Feature Characterization of Dislocation Networks. *SCRIPTA MATER* 240 (Feb. 2024), 115841.
[12] B. Katzer, K. Zoller, J. Bermuth, D. Weygand, and K. Schulz. 2023. Characterization of Lomer Junctions Based on the Lomer Arm Length Distribution in Dislocation Networks. *SCRIPTA MATER* 226 (March 2023), 115232.
[13] B. Katzer, K. Zoller, D. Weygand, and K. Schulz. 2022. Identification of Dislocation Reaction Kinetics in Complex Dislocation Networks for Continuum Modeling Using Data-Driven Methods. *J MECH PHYS SOLIDS* 168 (Nov. 2022), 105042.
[14] F. Li, Z. Zou, and J. Li. 2023. Durable Subgraph Matching on Temporal Graphs. *IEEE TKDE* 35, 5 (May 2023), 4713–4726.
[15] W.M. Lomer. 1951. A dislocation reaction in the face-centred cubic lattice. *Lond Edinb Dubl Phil Mag J Sci* 42, 334 (nov 1951), 1327–1331.
[16] K. Semertzidis and E. Pitoura. 2016. Durable Graph Pattern Queries on Historical Graphs. In *IEEE 32nd ICDE*. 541–552.
[17] K. Semertzidis and E. Pitoura. 2018. Top- k Durable Graph Pattern Queries on Temporal Graphs. *IEEE TKDE* 31, 1 (2018), 181–194.
[18] O. van Rest, S. Hong, J. Kim, X. Meng, and H. Chafi. 2016. PGQL: A Property Graph Query Language. In *Proc of 4th ACM GRADES*. 1–6.
[19] D. Weygand, L.H. Friedman, E. van der Giessen, and A. Needleman. 2001. Discrete dislocation modeling in three-dimensional confined volumes. *MAT SCI ENG A-STRUCT* 309-310 (jul 2001), 420–424.