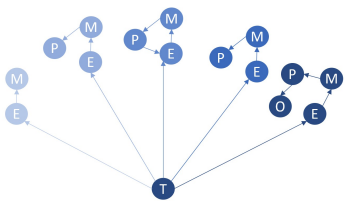
TEMPO

Management and Processing of Temporal Networks

H.F.R.I. Project No. 03480

D1.4: Second Year Project Report



Computer Engineering & Informatics Department
University of Patras
Greece
20/07/2024

Second Year Project Report of TEMPO

Kostas Christopoulos (PhD student), Alexandros Spitalas (PhD student), Kostas Tsichlas (PI)

July 20, 2024

Abstract

This document discusses the progress during the second year of the project. First, we report on the deliverables and milestones, as well as on the risks and deviations from the initial plan. Then, we discuss the work performed within each WorkPackage (WP) and report on dissemination activities.

1 Introduction

In the second year of the project, significant progress was made towards the intended goals. There was no change in the research team compared to the first year of the project. In particular, the research team consists of:

1. Konstantinos Tsichlas: Associate Professor at the University of Patras (Principal Investigator). He has a global view of the project and participates in all aspects of TEMPO.
2. Spyros Sioutas: Professor at the University of Patras. His main contribution is the provision of guidance related to graph-distributed databases.
3. Konstantinos Christopoulos: PhD student. He focuses on community detection (WP4) and outlier detection (WP5).
4. Alexandros Spitalas: PhD student. He focuses on the implementation of the historical graph management system (WP2) as well as on the implementation of the query engine for transactional and analytic queries (WP3).
5. Anastasion Gounaris: Professor in the Informatics Department of Aristotle University of Thessaloniki. His area of expertise is on distributed systems (among others), and he provided guidance related to query engines.
6. Apostolos Papadopoulos: Associate Professor in the Informatics Department of Aristotle University of Thessaloniki. His area of expertise is on graph databases (among others), and he provided guidance related to OLAP queries and related technologies (SPARK).

Overall, one deliverable (report) was prepared in the 2nd year of the project, while considerable progress in all WPs was achieved concerning the rest of the deliverables (demonstrations). In the remainder of this section, we give an overview of the progress in terms of deliverable preparation and milestone achievements. We also include a discussion regarding risks and deviations from the plan. In Section 2, we provide more details about the technical work in each WP. Finally, in Section 3, we provide an overview of the dissemination activities that took place in the second year of the project.

1.1 Deliverables

In the second year of the project, the following three deliverables were planned:

D1.4 2nd Year Project Report: The current document contains a description of the progress of TEMPO during the second year.

D2.1 Initial Version of the MAGMA System Prototype: This was intended as a first checkpoint highlighting the progress of the storage and maintenance mechanisms for the historical graph.

D3.1 Initial Version of the Query Engine Module: Similarly, this was intended as a first checkpoint highlighting the progress of the engine that will support a set of queries on the historical graph.

1.2 Milestones

There were two milestones anticipated for the second year: MS2 (planned at month 18) MS3 (planned at month 24). Both milestones are connected with deliverables D2.1 and D3.1 respectively. As discussed and thoroughly justified in the 1st Year Project Report document, there was a 6 month delay in achieving milestone MS1 that also affected the rest of the milestones. Milestone MS2 was affected the most since we had to make extensive adjustments in our scientific approach regarding the MAGMA system. Thus, milestone MS2 was achieved by M24 (a delay of 6 months). On the other hand, MS3 was achieved by the end of the 2nd year (M24) as anticipated. In 1.3, we discuss and justify these delays while in Section 2 we thoroughly discuss the progress for each WP. The progress related to milestones MS2 and MS3 are discussed in 2.2 and 2.3 respectively.

1.3 Risks and Deviation from the Plan

In the second year of the project, there is considerable progress in all WPs. Our main deviation from the initial plan is related to the system implementation (WP2). As thoroughly discussed and justified in the 1st Year Project Report, we decided instead of starting from scratch, to build on the open-source graph database JanusGraph to support efficiently the maintenance of the history of the graph. At the same time, we also decided to change the query engine Gremlin to support time-related queries on the graph database. On the one hand, we needed to familiarize ourselves with JanusGraph, which took us some time since we needed to see how JanusGraph combined with Apache Cassandra as the storage mechanism, could maintain historical graphs without making any changes to it (see 2.2). This resulted in achieving milestone MS2 on month 24. On the other hand, Gremlin allowed us to define and develop new queries pretty fast, thus easily achieving milestone MS3 (see 2.3).

2 Summary of Work per WP

A detailed report for each work-package follows.

2.1 WP1: Project Management - Result Dissemination

The main activity that took place was the preparation of the 2nd Year Project Report as well as 3 presentations at international conferences, as will be detailed in Section 3.

2.2 WP2: MAGMA System Implementation

During the 2nd year of the project, we familiarized ourselves with JanusGraph by studying its architecture and creating prototypes of the HiNode implementation using JanusGraph. While our ultimate goal is to modify JanusGraph to support the HiNode model, we began by implementing the HiNode model with JanusGraph combined with Apache Cassandra as the storage machine, using time as a simple property. This allowed us to experiment with JanusGraph and HiNode, using JanusGraph as a middleware instead of directly interacting with the storage model. However, we recognize that this approach is generally inefficient and not widely adoptable. Therefore, we plan to modify the core components of the JanusGraph API to integrate the HiNode model with the storage layer. This involves changing both the JanusGraph API to the storage layer (to follow the HiNode approach) and the JanusGraph API to the query language (to support time-constrained queries). Additionally, we need to add appropriate indexes in the index layer to efficiently support time queries.

In [14] we proposed a new implementation of the HiNode system using the Distributed-SQL Database CockroachDB. We tested this implementation in terms of a new algorithm to query Historical Degree Distribution in a node-centric Historical Graph. We tested HiNode in a distributed

SQL database since it is best suited for applications where data reliability is crucial. In addition, the new algorithm distributes query tasks and minimizes communication costs, making it ideal for applications where I/Os are expensive, CPU-intensive operations that shouldn't run locally, memory specifications are strict, or distributed machines have significantly more performance than the local machine. The goal of this work was for our team to get a grasp on how HiNode can be incorporated into a distributed SQL database and have a future reference for comparison within the JanusGraph environment.

In addition, we are preparing a journal submission that extends previous results on MongoDB, is also implements event-based transactions for a streaming environment and finally introduces a mechanism to create interval-based historical graph datasets that will be used to test our system. In summary, in this work, we review previous research on historical graph databases and storage systems, we introduce a way to generate historical graph datasets and we propose an enhanced methodology for a space-optimal vertex-centric model with MongoDB. This approach prioritizes space optimizations while managing to achieve notable advancements in the execution time of global queries, known for their complexity within entity-centric frameworks. Extensive experimentation has been done, employing snapshot-based and interval-based datasets generated with the use of the LDBC SNB generator, to validate our claims of achieving significant speed enhancements. Consequently, our approach enables the execution of more resource-intensive queries with improved efficiency, reduced client involvement, and less memory requirements.

Finally, we have made progress in partitioning Historical Graphs. Initially, we reviewed the existing partitioning methods for historical graphs and state-of-the-art techniques for static or dynamic graphs. Subsequently, we developed algorithms that are tailored to node-centric historical graphs. During our review, we discovered limited techniques for partitioning historical graphs, especially for vertex-centric historical graphs. It is challenging to directly apply state-of-the-art techniques for dynamic or static graphs to historical graphs. Indeed, we observed such an approach in one research study that has the extra overhead of first transforming the historical graph to a static graph. With these limitations in mind, we devised two algorithms suitable for a vertex-centric system. The first is an incremental partitioning algorithm designed to partition data as they are inserted into the database, while the second is a static partitioning algorithm that yields superior results but is more suitable for static historical graph databases. Finally, we conducted simulations to test these algorithms against the conventional hash-based approach, which has been the prevalent method for partitioning historical graphs thus far.

2.3 WP3: Complex Data Analytic Queries

Regarding queries, our final goal (which extends the goals of our initial plan) is to support both On-Line Transaction Processing (OLTP) and OnLine Analytical Processing (OLAP) historical queries in JanusGraph. These queries must be implemented in Gremlin as it is the main query language used in Janusgraph. Gremlin is a graph traversal query language as well as a graph traversal machine, designed and developed by Apache TinkerPop. It has been developed so that Gremlin traversal machine can be supported by a wide range of graph storage frameworks, including JanusGraph, Neo4j, Hadoop, and others. In the second year of the project, we first defined the semantics of the fundamental historical operations, and then we proposed the basic OLTP queries that a historical graph system should support. The following fundamental operations for inserting new nodes/edges/properties must be supported by the system.

InsertNode$(v, start, end)$ It creates a new empty historical node v with a valid interval $[start, end]$. If $start = -\infty$, then the node is valid for all time instances up to end. If $end = +\infty$, then the node is valid for all time instances after $start$. The same holds for all other operations as well, with respect to the use of the special symbols $\pm\infty$.

InsertEdge$(e = (u, v), start, end)$ It creates a new edge e with a valid interval $[start, end]$, which must be contained in the valid interval of both nodes u and v. The same check (although not mentioned) holds for the rest of the operations.

InsertProperty$(p, f, val, start, end)$ It creates a new property f in the historical object p (node or edge) with value val and a valid interval $[start, end]$.

The following operations are used to add or change the valid interval of a node/edge and to add values to an existing property.

ExtendNodeVI$(v, start, end)$ Node v has its valid interval extended by $[start, end]$. In case of intersection, we simply take the union.

ExtendEdgeVI$(e, start, end)$ Edge e has its valid interval extended by $[start, end]$. In case of intersection, we simply take the union.

ExtendProperty$(p, f, val, start, end)$ A new value val is added to the property f of the historical object p, such that val is valid in the time interval $[start, end]$. In case of intersections with other values of property f they are overwritten by appropriately managing the time intervals in the list of $p.f$.

The following operations are used to remove historical information from the historical graph.

ShrinkNodeVI$(v, start, end)$ Node v has its valid interval shrunk since it is invalidated in the time range $[start, end]$. This means that all properties of v and edges adjacent to v must be checked so that their valid intervals are still legitimate. In case they are not, then their valid intervals are shrunk as well by using other appropriate operations.

ShrinkEdgeVI$(e, start, end)$ Edge e has its valid interval shrunk, since it is invalidated in the time range $[start, end]$. This means that all properties of e must be checked so that their valid intervals are still legitimate. In case they are not, then their valid intervals are shrunk as well, by using other appropriate operations.

ShrinkPropertyVI$(p, f, start, end)$ The valid interval of property f of the historical object p is shrunk, since it is invalidated in the time range $[start, end]$.

Using the metaprogramming provided by the Groovy language, a modification or addition of methods to a class or interface can be executed to Gremlin. Thus, we developed the following methods in Gremlin (that implement the previously mentioned operations) using the Groovy metaprogramming. In the current implementation, these methods need to be imported into Gremlin so that they can be supported out of the box. These methods are shown in Table 1 and they can be used by just adding a patch to Gremlin during its initialization phase.

lifetime(start, end)	Stores as property of the vertex/edge its active period
vid(value)	Stores as property of the vertex the given id
insertE(label, sourceid, targetid)	Inserts a labeled edge between the given vertices
weight(value)	Stores as property of the edge the given value
addA(name, value , start, end)	Stores an attribute as a property of the vertex and its active period as meta-properties
deleteV(vid, end)	Sets the termination of the vertex with the specified vid at the specified time
deleteE(sourceid, targetid, end)	Sets the termination of the defined edge at the defined time

Table 1: Custom Gremlin methods.

Historical OLAP queries have not yet been implemented in Gremlin, but they can be executed either using the Janusgraph internal API, or the Gremlin API, but in both cases the user must change the queries accordingly.

2.4 WP4: Local Community Detection

During the 2nd year of the project our research efforts were directed toward developing new variants of an existing local community detection algorithm. The objective was to enhance the detection of

single communities around significant seed nodes, ensuring faster and more accurate subgroup identification compared to global methods. Extensive experiments were conducted on both synthetic and real datasets. The new methods demonstrated superior performance over a baseline algorithm, three state-of-the-art local community detection methods, and one global community detection method. These results have been compiled into a research paper, highlighting the significant advancements in local community detection and its applications in network analysis.

In the third year of the project, the focus will be on developing a distributed algorithm in Spark, based on [2], for community detection within specified query time intervals in historical graphs. The method will identify communities by evaluating the contributions of each edge/node during the user-defined time interval. This novel approach, not previously addressed in the literature, aims to enhance the understanding of evolving connections in static networks with temporal dimensions. As soon as a working version of the system is available, these algorithms will be implemented on this system with the help of Gremlin.

2.5 WP5: Outlier Detection

This year, recent developments in community-based anomaly detection in dynamic networks were comprehensively examined. An established algorithm for dynamic local community detection was introduced, aimed at identifying anomalies based on community evolution over time. Preliminary experimental findings using synthetic datasets were provided to support the study.

In the third year of the project, the focus will be on developing a distributed algorithm in Spark for anomaly detection in dynamic networks. The algorithm will analyze static aggregate community detection within specified query time intervals in historical graphs, evaluating the contributions of each edge/node during the user-defined time interval. As soon as a working version of the system is available, these algorithms will be implemented on this system with the help of Gremlin.

3 Publications and Other Dissemination Material

There are eight publications (six conferences and two journals) from the research work conducted until July 2024:

1. **State-of-the-art in Community Detection in Temporal Networks** [8] that contains the work within deliverable D4.1. (*1st Year*)
2. **MAGMA: Proposing a Massive Historical Graph Management System** [13] that contains part of the survey within deliverable D1.2 as well as a preliminary version of the system architecture of MAGMA. (*1st Year*)
3. **Dynamic Local Community Detection with Anchors** [4] that contains a streaming algorithm for local community detection by using anchors instead of seeds. (*1st Year*)
4. **Local Community Detection: A survey** [5] that contains a survey on local community detection algorithms based partially on deliverable D4.1. Note that no such stand-alone survey existed for the problem of local community detection. (*1st Year*)
5. **Local Community Detection in Graph Streams with Anchors** [9] that contains extended research on local community detection in a streaming environment based on [4]. (*1st Year*)
6. **Adopting Different Strategies for Improving Local Community Detection: A Comparative Study** [3] that introduce new variants of an existing local community detection algorithm that uncover a single community. (*2nd Year*)
7. **Local Community-Based Anomaly Detection in Graph Streams** [1] that introduces an established algorithm for dynamic local community detection, aimed at identifying anomalies based on community evolution over time. (*2nd Year*)
8. **Degree Distribution Optimization in Historical Graphs** [14] that builds HiNode model on top of CockroachDB, a Distributed SQL database, for more resilience of data, and also introduces a new space-efficient algorithm for degree distribution query in historical graphs. (*2nd Year*)

References

[1] **Local Community-Based Anomaly Detection in Graph Streams.** Konstantinos Christopoulos, and Konstantinos Tsichlas. *In Proc. of the 20th International Conference on Artificial Intelligence Applications and Innovations*, 2024.

[2] **High quality, scalable and parallel community detection for large real graphs.** Prat-Pérez, Arnau, David Dominguez-Sal, and Josep-Lluis Larriba-Pey. *In Proc. of the 23rd international conference on World wide web*, 2015.

[3] **Adopting Different Strategies for Improving Local Community Detection: A Comparative Study.** Konstantinos Christopoulos, and Konstantinos Tsichlas. *In Proc. of the 12th Intern. Conference on Complex Networks and their Applications*, 2023.

[4] **Dynamic Local Community Detection with Anchors.** G. Baltsou, K. Christopoulos and K. Tsichlas. *In Proc. of the 11th Intern. Conference on Complex Networks and their Applications*, 2022.

[5] **Local Community Detection: A survey.** G. Baltsou, K. Christopoulos and K. Tsichlas. *IEEE ACCESS*, vol. 10, pp. 110701-110726, 2022.

[6] **Local community detection with hints.** G. Baltsou, K. Tsichlas and A. Vakali. *Applied Intelligence*,doi:10.1007/s10489-021-02946-7, 2022.

[7] **Dynamic Community Detection with Anchors (Extended Abstract)** G. Baltsou and K. Tsichlas. *In Proc. of the 10th Intern. Conference on Complex Networks and their Applications*, 2021.

[8] **State-of-the-art in Community Detection in Temporal Networks.** K. Christopoulos and K. Tsichlas. *In Proc. of the 18th International Conference on Artificial Intelligence, Applications, and Innovations (AIAI) - Mining Humanistic Data Workshop (MHDW)*, 2022.

[9] **Local Community Detection in Graph Streams with Anchors.** K. Christopoulos, G. Baltsou and K. Tsichlas. *Information*, 14(6): 332, 2023.

[10] **HiNode: An Asymptotically Space-Optimal Storage Model for Historical Queries on Graphs.** A. Kosmatopoulos, K. Tsichlas, A. Gounaris, S. Sioutas and E. Pitoura. *Distributed and Parallel Databases*, 35(3-4):249–285, 2017.

[11] **Hinode: Implementing a Vertex-Centric Modelling Approach to Maintaining Historical Graph Data.** A. Kosmatopoulos, A. Gounaris and K. Tsichlas. *Computing*, pp. 1-24, 2019.

[12] **Investigation of Database Models for Evolving Graphs.** A. Spitalas, A. Gounaris, A. Kosmatopoulos, K. Tsichlas. *In proc. of the 28th International Symposium on Temporal Representation and Reasoning(TIME)*, pp. 6:1-6:13, 2021.

[13] **MAGMA: Proposing a Massive Historical Graph Management System.** A. Spitalas and K. Tsichlas. *In Proc. of the 7th International Symposium on Algorithmic Aspects of Cloud Computing ALGOCLOUD*, pp. 42-57, 2022.

[14] **Degree Distribution Optimization in Historical Graphs.** A. Spitalas, Charilaos Kapeletiotis, and K. Tsichlas. *Submitted to the 9th International Symposium on Algorithmic Aspects of Cloud Computing ALGOCLOUD.*