SoftwareX 26 (2024) 101668

Contents lists available at ScienceDirect

SoftwareX

journal homepage: www.elsevier.com/locate/softx

Original software publication

phyloDB: A framework for large-scale phylogenetic analysis of sequence based typing data

Bruno Lourenço [a], Cátia Vaz [b,c,*], Miguel E. Coimbra [c], Alexandre P. Francisco [a,c]

[a] Instituto Superior Técnico, Universidade de Lisboa, Portugal
[b] Instituto Superior de Engenharia de Lisboa, Instituto Politécnico de Lisboa, Portugal
[c] INESC-ID Lisboa, Portugal

ARTICLE INFO

Keywords:
Large scale phylogenetic analysis
Phylogenetic inference
Graph database

ABSTRACT

phyloDB is a modular and extensible framework for large-scale phylogenetic analyses of sequence based typing data, which are essential for understanding epidemics evolution. It relies on the Neo4j graph database for data storage and processing, providing a schema and an API for representing and querying phylogenetic data. Custom algorithms are also supported, allowing users to perform heavy computations directly over the data, and to store results in the database. Multiple computation results are stored as multilayer networks, promoting and facilitating comparative analyses, as well as avoiding unnecessary ab initio computations. The experimental evaluation results showcase that phyloDB is efficient and scalable with respect to both API operations and algorithms execution.

Code metadata

Current code version	v1.2.2
Permanent link to code/repository used for this code version	https://github.com/ElsevierSoftwareX/SOFTX-D-23-00619
Permanent link to Reproducible Capsule	n/a
Legal Code License	GNU GPL v3
Code versioning system used	git
Software code languages, tools, and services used	Java, Neo4j
Compilation requirements, operating environments & dependencies	JDK 11, Spring Boot, jackson-core, Neo4j, Gradle, Apache Maven
If available Link to developer documentation/manual	https://github.com/phyloviz/phyloDB
Support email for questions	cvaz@cc.isel.ipl.pt; aplf@inesc-id.pt

1. Motivation and significance

Modern biomedical research and engineering has seen a remarkable increase in the production and computational analysis of large datasets, leading to an urgent need of efficient and scalable tools for data integration, processing and visualization, as well as the need of sharing standardized analytical techniques. One important field of biomedical research is understanding the evolution of pathogens in order to determine their origin, evolution and resistance. For instance, phylogenetic analyses have been essential in foodborne pathogen surveillance [1].

When performing large-scale phylogenetic analyses of microbial population genetics, it is often needed to sequence and type isolates, and afterwards to apply a set of phylogenetic inference methods [2] to produce a diagrammatic hypothesis about the evolutionary history. The computation and analysis of microbial population genetics often produces phylogenetic trees or networks [3]. The appearance of NGS technologies further increased this challenge due to the substantial growth in the amount of genomic and typing data that can be used to characterize a population.

The integration of the results obtained from inference algorithms with epidemiological data (also called isolate ancillary data) and simultaneous analysis is still limited by visualization and processing techniques. Although there are some tools for visualizing and analyzing such data, allowing the integration of epidemiological data, such as SplitsTree4 [4], Phylogeny.fr [5], PHYLOViZ [6,7] and GrapeTree [8],

* Corresponding author at: Instituto Superior de Engenharia de Lisboa, Instituto Politécnico de Lisboa, Portugal.
E-mail address: cvaz@cc.isel.ipl.pt (Cátia Vaz).

https://doi.org/10.1016/j.softx.2024.101668
Received 17 September 2023; Received in revised form 5 February 2024; Accepted 18 February 2024
Available online 22 February 2024

they do not scale for large data analysis and visualization. Most of them run inference and/or visualization optimization tasks on the client side, requiring data to be transferred from existing databases in order to be analyzed. For instance, although PHYLOViZ allows to connect to public data repositories and databases, data must be retrieved and processed locally. Even when some of these tools are deployed together with a database, e.g., Enterobase [9] includes a deployment of GrapeTree, data must be retrieved from the database and loaded in the tool for phylogenetic analysis. This is often unfeasible for large amounts of typing data and, moreover, results and optimizations are not stored for reuse and updating. An illustrative example of such large datasets is the *Salmonella* database [10], present in Enterobase, that has more than 450,000 strains and where data has been growing linearly with time.

Graph databases, such as Neo4j [11], address the problem of leveraging complex and dynamic relationships in highly connected data such as the phylogenetic trees or networks. When the storage is native, such as in Neo4j, their performance tends to remain relatively constant, in contrast to the traditional relational databases, even as the dataset grows. The reason for this performance in graph databases is related to the fact that queries are local to a portion of the graph, and thus execution time for each query is proportional to the size of that part. In terms of flexibility, a graph database supports the addition of new nodes, labels and relationships to an existing structure, without jeopardizing existing queries and application functionality. These features are very important for phylogenetic context, since a native graph storage is optimized and designed for storing and managing graphs, and with a native processing engine, query times are proportional to the amount of the graph data searched (independent of the total size of the graph).

With respect to semantic extensibility, Neo4j, for instance, provide mechanisms that allow the extension of the database semantics, by implementing custom functions or procedures. This is crucial in our setting, enabling the avoidance of data transfer and loading for phylogenetic analyses of sequence based typing data, and allowing for more rich algorithms to be implemented and efficiently deployed over the data.

Although graph-oriented databases can be of much help for biomedical data [12,13], as far as we know there is no solution relying on these technologies to address large-scale phylogenetic analysis challenges.

Thus, the contribution of this work is PHYLODB, a modular framework for large-scale phylogenetic analyses of sequence based typing data, exploiting the Neo4j graph-oriented database to allow for the management of the phylogenetic data, without needing to load it into client computers, or secondary servers, for further exploration. PHYLODB has a graph data model that allows the representation of phylogenetic data, in particular microbial sequence based typing data, inferred phylogenetic trees and networks through distance based inference algorithms [14], as well as related ancillary data. It efficiently supports queries over such data, and allows the deployment of algorithms for inferring/detecting patterns of evolution and for pre-computing and optimizing phylogenetic trees visualizations. These inference and optimization tasks can be requested through the API by submitting jobs to the server, checking their state, and retrieving the results when ready. We note however that PHYLODB allows also to fully implement dynamic algorithms that update inference patterns as new data becomes available [15], since dynamic algorithms can be deployed as procedures and triggered on demand; this is crucial for large scale analyses. By storing the results of algorithms with the data, it also enables the reuse and comparison of results.

Finally, we note that PHYLODB can be integrated with other tools through its API. For instance, since phylogenetic trees are provided in Newick format through the API, PHYLODB can be integrated with Taxonium [16] for visualizing and exploring large phylogenetic trees.

2. Software description

PHYLODB is a framework that allows users to store and manage data resulting from phylogenetic analyses in a Neo4j graph database. This framework supports the execution of inference and visualization algorithms directly over the stored data offloading heavy computations to the server side. This is possible by supporting algorithms as Neo4j plugins, i.e., user-defined procedures in Neo4j. A user-defined procedure is a mechanism that allows to extend Neo4j by writing custom code, which can be invoked directly from its query language Cypher. PHYLODB provides also a secured API for phylogenetic data management.

2.1. Software architecture

This framework consists of several components as depicted in Fig. 1. The server component provides a Spring [17] web application programming interface (API) to perform several operations over the data stored in the database, namely data access and loading, the execution of algorithms and retrieval of results. It can scale horizontally by adding more instances. This component interacts with the database component, namely with the Neo4j graph database, for data storage, data management and for queuing the execution of algorithms deployed as plugins. These algorithms can also read data from the graph database, and write back computed results. The authorization component manages the user information and validates the operations available to the authenticated user. Although we can use any authentication provider, the present implementation relies on Google Identity Provider [18] and on a Keycloak Provider [19] for authentication.

2.1.1. Data storage

We rely on a graph data model built from a subset of the concepts and properties defined in the TypOn ontology [20], allowing the representation of the main entities in phylogenetic analyses of sequence based typing data, such as multilocus sequence typing (MLST), core genome MLST, whole genome MLST or single-nucleotide polymorphism (SNP) data, specially for bacterial identification and characterization at subspecies level, as well as their relationships (see Fig. 2). Other entities such as users and projects were introduced to support user and project management, including authorization and data versioning. Moreover, we also have introduced entities relevant to store results from inference and visualization algorithms. For instance, considering distance-based inference methods, one common step of these methods is the calculation of a distance matrix that reflects the pairwise genetic distance between nodes of the phylogenetic tree or network. In the phylogenetic context, nodes represent profiles, i.e, characterizations of the genetic sequences of isolate data. Given a distance matrix, an inference algorithm is executed and links among nodes can be created. Since links are labeled accordingly to the id of the performed inference, different inference results can be registered, and we get a multilayer network among nodes.

Visualizations of inferred patterns represented by links are optimized and rendered through methods such as the Radial Layout [21] or GrapeTree [8]. The optimization process may take considerable time, and usually it can be shared and reused. For instance, the optimized layout computation of a phylogenetic tree for the *Salmonella* database [10] present in Enterobase [9], with more than 450,000 strains, is not actually feasible on the client side, both in what concerns memory and time, taking several hours. This motivates the interest in separating the computation of the optimal coordinates for each node in the visualization from the rendering process. PHYLODB supports the execution of visualization optimizations over the data on the server side, and it stores obtained coordinates in the database for each different layout algorithm execution. Precomputed visualizations can then be loaded by client applications, and further optimized and manipulated by users on client applications.

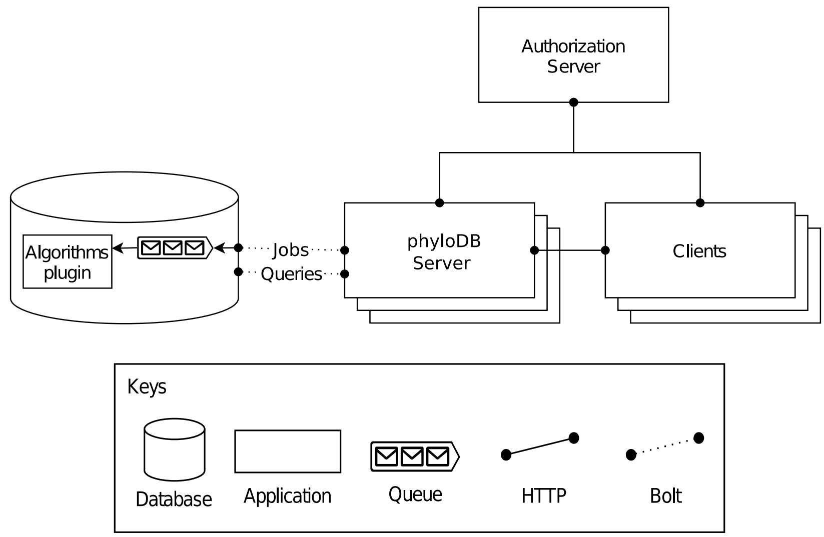

Fig. 1. Client–Server architectural view.

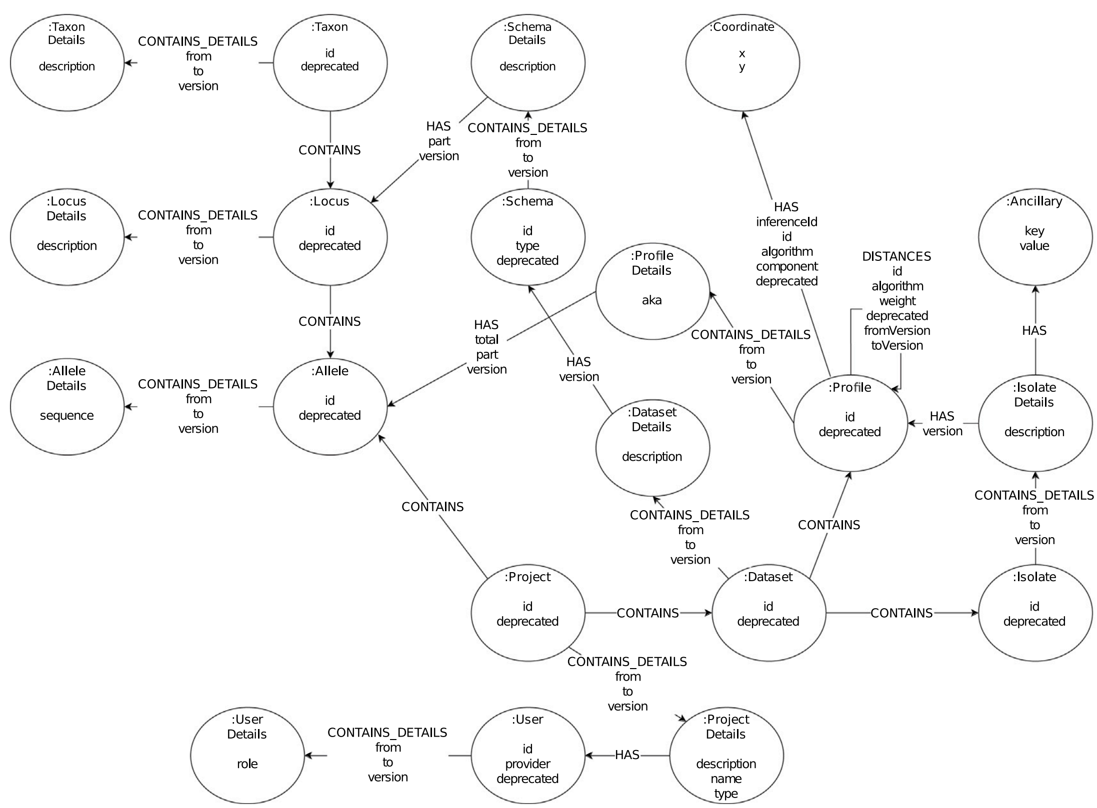

Fig. 2. Data model represented by a set of nodes and relationships to compose a graph data model, which is the format used by Neo4j.

The data model also supports versioning and soft deletion. Given the data dependencies, the importance of keeping track of changes and for the sake of reproducible results, we avoid deleting information in the database. In this case, by considering a versioning and a soft-delete strategy, information removal is possible while keeping previous results valid for the underlying version of the data. The versioning strategy to achieve this behavior is to separate each object from its state, link them through a relationship with the respective version number, and capture changes by having different state nodes [22]. Data soft-deleted are marked deprecated and, hence, become unusable for new analyses. API results do not include also deprecated data unless explicitly requested. Users can however access and explore previous inference results that may include deprecated data, which is marked as such. The data model is detailed in the phyloDB repository https://github.com/phyloviz/PhyloDB/wiki/Labeled-Property-Graph-Model.

2.2. Software functionalities

The main functionalities of phyloDB are (1) a secured API to perform data management operations over the data stored in the graph database, such as data access, querying, loading, and the execution of inference and visualization optimization tasks, storing their results; (2) data versioning; and (3) a plugin interface for adding new inference and visualization algorithms.

2.2.1. Web API

The API was designed not only to allow operations over the data stored in the graph database, but also to support algorithms execution and to provide an easy integration of the framework with other applications such as front-end applications or workflow systems.

The API provided by PHYLODB was implemented based on three layers, namely `Controllers`, `Services` and `Repositories`. When a request is received by the API, it is passed through the `Controllers` layer. This layer contains the controllers that parse the received input, execute the respective service, and retrieve the response containing the respective status code and the formatted content. The `Services` layer performs the business logic and depends on operations over repositories. The `Repositories` layer makes available the operations to interact with the database. Apart from these layers, there is a validation logic that verifies the request authenticity and the user permissions before the request is processed. The API relies on the REST architecture, and on the HTTP semantics. The set of endpoints that were defined are described in the PHYLODB repository https://github.com/phyloviz/phyloDB/wiki/API-documentation. Examples of the API usage are also available at https://github.com/phyloviz/phyloDB/blob/v1.2.2/scripts/example.

The authentication is based on bearer token authentication [23]. This type of authentication is based on tokens that are acquired after an authentication process with an identity provider.

We note that several aspects were considered in the implementation of repositories, such as the use of an object graph mapper (OGM), pagination and parameterized queries, among others. The use of the functionalities related to an OGM may add some overhead [24]. Hence, it was decided that each query should be implemented from scratch to increase the performance of the data access operations. The implementation of each repository method contains then the respective `Cypher` query that it should perform. Moreover, the use of pagination in these queries was adopted because this framework is intended to handle large quantities of data. Parameterized queries are also used so for better performance, as Neo4j can cache the query plans and reuse them in subsequent executions, increasing the speed of the next query. This also enables protection against injection attacks, since parameters are never allowed to be interpreted as part of the query and have no means of escaping out of being anything other than a value of some sort [24,25].

2.2.2. Algorithms as plugins

The plugin functionality relies on the user-defined procedures from Neo4j and on the APOC library [26]. A user-defined procedure is a mechanism that extends Neo4j through custom code. These procedures can take arguments, perform operations on the database and return results. They can be also invoked directly from `Cypher`.

In our work plugins intend to extend Neo4j to support inference and visualization algorithms, which shall be available as procedures. The inference algorithm procedures are executed over the profiles of a dataset, while visualization algorithm procedures are executed over the results of inference algorithms. The fact of being possible to run the algorithms directly on the graph database is an important feature since datasets are growing daily. And, when using distance based phylogenetic inference algorithms, it is necessary to calculate pairwise genetic distances. This step takes $O(n(n+m))$ space and $O(n^2m)$ time, with n being the number of nodes (allelic profiles) and m the size of each allelic profile. Moreover, some distance based inference algorithms, such as Neighbor Joining [27] and variants, need another matrix of a similar size during its execution. Taking the *Salmonella* database as an example, with more than 450,000 allelic profiles and growing, and with wgMLST allelic profiles with 21,000 loci, it becomes clear that running these inference methods is becoming unfeasible on the client side. PHYLODB makes it possible also to fully implement dynamic algorithms that update inference patterns as new data becomes available [15].

The structure of an algorithm plugin is also based on three layers, namely (1) the `Procedures` layer that provides the fundamental operations needed for user-defined algorithms; (2) the `Services` layer responsible for reading input data from the database, executing algorithms, and storing results back on the database; (3) and the `Repositories` layer that provides methods to interact with the database. Once implemented as described below, an algorithm can be compiled and packaged in a jar file, and deployed in the Neo4j `plugins` directory together with other plugins and the APOC library.

3. Illustrative examples

We illustrate in this section the addition of a new algorithm. Further details and examples are provided in the project documentation, including the API documentation and use cases.

Let us add goeBURST [28] algorithm as an example of an inference algorithm. In this context, it must be taken the following steps: (1) create a new procedure; (2) create a new service; (3) create a new repository; (4) compile and package in a jar file. See the project repository for the full example, https://github.com/phyloviz/phyloDB/tree/master/algorithms/src/main/java/algorithm/inference.

3.1. New procedure

Different procedures can be added by creating sub-types of the procedure type. As depicted in Listing 1, `Procedure` is defined as an abstract type that provides the common base for all procedures.

Listing 1: Procedure.

```
public abstract class Procedure {
  @Context
  public GraphDatabaseService database;
  @Context
  public Log log;
}
```

In Listing 2, for inference algorithms, a new sub-type is defined. Methods declared within this new type must be annotated with the `@Procedure` annotation to be executed as a standard procedure. Also, with this annotation, the designation of the new standard procedure is defined. In the example of Listing 2, to execute the new standard procedure `goeBURST`, the designation to use is `algorithms.inference.goeburst`.

Listing 2: InferenceProcedure.

```
public class InferenceProcedure extends algorithm.utils.Procedure {
  /*...*/
  @Procedure(value = "algorithms.inference.goeburst",
             mode = Mode.WRITE)
  public void goeBURST(@Name("project") String project,
        @Name("dataset") String dataset,
        @Name("lvs") long lvs,
        @Name("inference") String inference) {

    InferenceService service = new InferenceService(database, log);
    service.goeBURST(project, dataset, inference, lvs);
  }
}
```

3.2. New service

Listing 3 shows the common abstract class `Service`. Considering the goeBURST example, the `InferenceService` type is defined as a sub-class of `Service`, executing all steps required to run the goeBURST algorithm, as depicted in Listing 4.

Listing 3: Service.

```
public abstract class Service {
  public GraphDatabaseService database;
  public Log log;
```

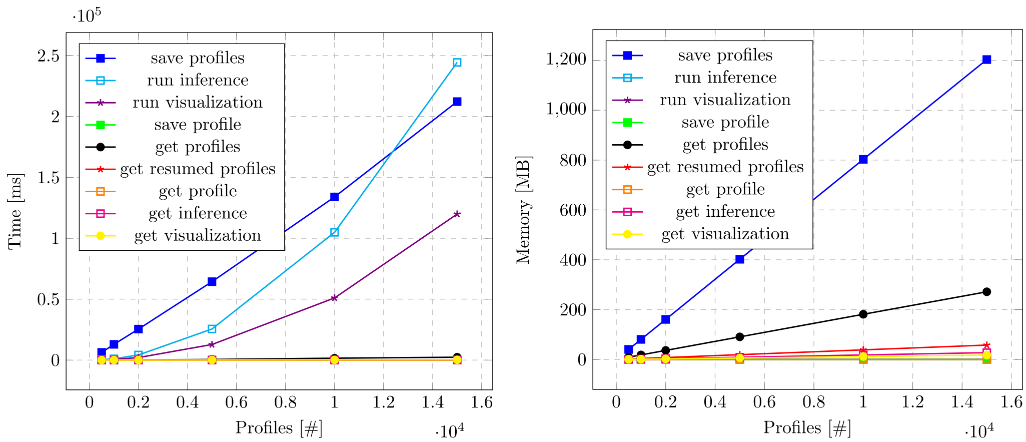

Fig. 3. Plots containing the time and the memory results, in milliseconds and megabytes respectively, as new profiles are incrementally integrated for the *Streptococcus pneumoniae* MLST dataset.

```
  public Service(GraphDatabaseService database, Log log) {
    this.database = database;
    this.log = log;
  }
}
```

Listing 4: InferenceService.

```
public class InferenceService extends Service {
/*...*/
  public void
    goeBURST(String project, String dataset, String analysis, long lvs){
      InferenceRepository repository = new InferenceRepository(database);
      GoeBURST algorithm = new GoeBURST();
      algorithm.init(project, dataset, analysis, lvs);
      Matrix matrix;
      try (Transaction tx1 = database.beginTx()) {
        matrix = repository.read(tx1, project, dataset);
        tx1.commit();
      }
      Inference inference = algorithm.compute(matrix);
      try (Transaction tx2 = database.beginTx()) {
        repository.write(tx2, inference);
        tx2.commit();
      }
  }
}
```

The goeBURST algorithm is implemented in `GoeBURST`, which implements the helper interface `Algorithm`. This interface defines the minimal set of operations that an algorithm should support.

3.3. New repository

The `Repository` abstract type is provided in order to define the common read and write methods, as partially depicted in Listing 5.

Listing 5: Repository.

```
public abstract class Repository<T, R> {
  /*...*/
  public abstract R read(Transaction tx, String... params) throws
  Exception;
  public abstract void write(Transaction tx, T param);
}
```

Then, for the goeBURST algorithm example, it is also necessary to implement the `InferenceRepository` type, sub-type of `Repository` to provide the implementation of these methods for this algorithm.

3.4. Jar package

If the implementation just described is added to the existing algorithms library, a new version of this library can be created by following the build instructions explained in the wiki of the project. It is also possible to define them as an extension to the algorithms library, producing a new separated jar as usual. In both cases, the new library jar must be added to the plugins directory of Neo4j along with required dependencies.

4. Impact

PHYLODB is a modular framework that provides the foundations to efficiently manage, process and analyze sequence based typing data in the context of large phylogenetic studies. This is accomplished by making available a secured API and by supporting new algorithms through plugins that can be executed directly on the underlying graph database. An important feature is that new algorithms can be implemented, modified and deployed without interfering with the API and graph database functionality or availability. PHYLODB stores and makes available not only phylogenetic typing data, but also results from inference algorithms and optimized visualization information. This allows the reuse and comparison of inference results, readily available with the underlying typing data, avoiding the repeated execution of inference algorithms. As described in the project repository, PHYLODB can be easily deployed and managed through container orchestration, with the API being horizontally scalable.

PHYLODB is currently being integrated within PHYLOViZ Online [7]. It has permitted to offload optimization tasks from the client side, and to deal with larger datasets. Results concerning the performance of PHYLODB on real data is available in Fig. 3. The tests were executed several times over a set of read and write operations, over the *Streptococcus pneumoniae* MLST dataset [29], which was specifically chosen because it is part of several published studies and also because it is publicly available, which will facilitate the interpretation and reproducibility of the results. Note that inference and visualization optimization operations not only include the execution of the algorithm but also the work of gathering the data and storing the results, which depends on several transactions in the underlying database. Obtained results confirm that the operations that deal with a fixed amount of data are not affected by the increasing volume of the data stored. Additionally, we note that execution times for the algorithms comply with their expected time complexity. This showcases that relying on a graph database to handle this type of data enables good performance and scalability.

PHYLODB allows also now to make available algorithms and methods for the dynamic updating of inferred phylogenetic trees and networks, as well as visualizations. Although previous work is known [15], such approach is only applicable in practice if algorithms are possible to run in a setting as provided by PHYLODB. Future work comprises then the development and implementation of algorithms supporting dynamic updating of phylogentic inferences and visualizations, fundamental for large scale analyses.

5. Conclusions

Epidemics have become an issue of increasing importance [30] due to the growing exchanges of people and merchandise between countries. Hence, phylogenetic analyses are continuously generating huge volumes of typing and ancillary data. There is no doubt about the importance of such data, and phylogenetic studies, for the surveillance of infectious diseases and the understanding of pathogen population genetics and evolution. And the traditional way of performing phylogenetic analyses for sequence based typing data is becoming unfeasible given the amount of data generated. The goal of this work was to develop a foundational framework for computational phylogenetic analyses that exploits graph databases with traits such as those of Neo4j.

The implementation of this framework provides a data model that is designed to represent the relationships between the several types of data and to consider multilayer networks, which enable superimposing multiple inference results and visualizations. This data model also contemplates versioning and soft-delete operations, allowing to keep the history of evolving datasets and analyses. Furthermore, the API implementation considers several other requirements, such as importing and exporting of datasets, logging, error handling and security. Finally, the implementations of algorithms are based on the user-defined procedures feature of Neo4j, which allows the extension of its semantic with new algorithm plugins.

CRediT authorship contribution statement

Bruno Lourenço: Conceptualization, Software. **Cátia Vaz:** Conceptualization, Software, Supervision, Validation, Writing – original draft, Writing – review & editing. **Miguel E. Coimbra:** Software, Validation, Writing – original draft. **Alexandre P. Francisco:** Conceptualization, Project administration, Supervision, Validation, Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

The data used is publicly available, as explained in the manuscript.

Acknowledgments

The work reported in this article as received funding from the European Union's Horizon 2020 research and innovation program under Grant Agreement No. 951970 (OLISSIPO project). It was also supported by national funds through Fundação para a Ciência e a Tecnologia (FCT) with reference UIDB/50021/2020 (http://dx.doi.org/10.54499/UIDB/50021/2020).

References

[1] Timme RE, Strain E, Baugher JD, Davis S, Gonzalez-Escalona N, Sanchez Leon M, et al. Phylogenomic pipeline validation for foodborne pathogen disease surveillance. J Clin Microbiol 2019;57(5):10–128.

[2] Vaz C, Nascimento M, Carriço JA, Rocher T, Francisco AP. Distance-based phylogenetic inference from typing data: a unifying view. Brief Bioinform 2020. http://dx.doi.org/10.1093/bib/bbaa147.

[3] Teixeira AS, Monteiro PT, Carriço JA, Ramirez M, Francisco AP. Not seeing the forest for the trees: size of the minimum spanning trees (MSTs) forest and branch significance in MST-based phylogenetic analysis. PLoS One 2015;10(3):e0119315.

[4] Huson DH, Bryant D. Application of phylogenetic networks in evolutionary studies. Mol Biol Evol 2006;23(2):254–67.

[5] Dereeper A, Guignon V, Blanc G, Audic S, Buffet S, Chevenet F, et al. Phylogeny. fr: robust phylogenetic analysis for the non-specialist. Nucleic Acids Res 2008;36(suppl_2):W465–9.

[6] Nascimento M, Sousa A, Ramirez M, Francisco AP, Carriço JA, Vaz C. PHYLOViZ 2.0: providing scalable data integration and visualization for multiple phylogenetic inference methods. Bioinformatics 2016;33(1):128–9.

[7] Ribeiro-Gonçalves B, Francisco AP, Vaz C, Ramirez M, Carriço JA. PHYLOViZ Online: web-based tool for visualization, phylogenetic inference, analysis and sharing of minimum spanning trees. Nucleic Acids Res 2016;44(W1):W246–51.

[8] Zhou Z, Alikhan N-F, Sergeant MJ, Luhmann N, Vaz C, Francisco AP, et al. GrapeTree: visualization of core genomic relationships among 100,000 bacterial pathogens. Genome Res 2018;28(9):1395–404.

[9] Zhou Z, Alikhan N-F, Mohamed K, Fan Y, Achtman M, Brown D, et al. The EnteroBase user's guide, with case studies on Salmonella transmissions, yersinia pestis phylogeny, and Escherichia core genomic diversity. Genome Res 2020;30(1):138–52.

[10] Alikhan N-F, Zhou Z, Sergeant MJ, Achtman M. A genomic overview of the population structure of Salmonella. PLoS Genet 2018;14(4):e1007261.

[11] Holzschuher F, Peinl R. Performance of graph query languages: Comparison of cypher, gremlin and native access in Neo4J. In: Proceedings of the joint EDBT/iCDT 2013 workshops. New York, NY, USA: ACM; 2013, p. 195–204. http://dx.doi.org/10.1145/2457317.2457351.

[12] Sakr S, Pardede E. Graph data management: techniques and applications. Information science reference-imprint of, US: IGI Publishing; 2011.

[13] Yoon B-H, Kim S-K, Kim S-Y. Use of graph database for the integration of heterogeneous biological data. Genom Inform 2017;15(1):19–27.

[14] Vaz C, Nascimento M, Carriço JA, Rocher T, Francisco AP. Distance-based phylogenetic inference from typing data: a unifying view. Brief Bioinform 2021;22(3):bbaa147.

[15] Francisco AP, Nascimento M, Vaz C. Dynamic phylogenetic inference for sequence-based typing data. In: Proceedings of the 8th ACM international conference on bioinformatics, computational biology, and health informatics. 2017, p. 604–604.

[16] Sanderson T. Taxonium, a web-based tool for exploring large phylogenetic trees. Elife 2022;11.

[17] Walls C. Spring in action. US: Simon and Schuster; 2022.

[18] Google. OpenIDConnect. 2023, URL https://developers.google.com/identity/protocols/OpenIDConnect. [Last accessed 29 January 2023].

[19] Thorgersen S, Silva PI. Keycloak - identity and access management for modern applications: harness the power of keycloak, openID connect, and OAuth 2.0 protocols to secure applications. Packt Publishing; 2023.

[20] Vaz C, Francisco AP, Silva M, Jolley KA, Bray JE, Pouseele H, et al. TypOn: the microbial typing ontology. J Biomed Semant 2014;5(1):43. http://dx.doi.org/10.1186/2041-1480-5-43.

[21] Bachmaier C, Brandes U, Schreiber F. Biological networks. 2014.

[22] Lazarevic L. Keeping track of graph changes using temporal versioning. 2023, URL https://medium.com/neo4j/keeping-track-of-graph-changes-using-temporal-versioning-3b0f854536fa. [Last accessed 29 January 2023].

[23] Jones M, Hardt D. The OAuth 2.0 authorization framework: Bearer token usage. 2023, URL https://tools.ietf.org/html/rfc6750. [Last accessed 29 January 2023].

[24] Willemsen C. Cypher: Write fast and furious. 2023, URL https://neo4j.com/blog/cypher-write-fast-furious/. [Last accessed 29 January 2023].

[25] Bowman A. Neo4j security. 2023, URL https://community.neo4j.com/t/neo4j-security/16044. [Last accessed 29 January 2023].

[26] Neo4j. APOC user guide for neo4j v5. 2023, URL https://neo4j.com/docs/apoc/. [Last accessed 04 February 2023].

[27] Saitou N, Nei M. The neighbor-joining method: a new method for reconstructing phylogenetic trees. Mol Biol Evol 1987;4(4):406–25.

[28] Feil EJ, Li BC, Aanensen DM, Hanage WP, Spratt BG. eBURST: inferring patterns of evolutionary descent among clusters of related bacterial genotypes from multilocus sequence typing data. J Bacteriol 2004;186(5):1518–30.

[29] Jolley K, Bray J, Maiden M. Open-access bacterial population genomics: BIGSdb software, the PubMLST. org website and their applications. Wellcome Open Res 2018;3(124). URL http://pubmlst.org/.

[30] Surveillances V. The epidemiological characteristics of an outbreak of 2019 novel coronavirus diseases (COVID-19)—China, 2020. China CDC Wkly 2020;2(8):113–22.