MULTI-RESOLUTION REPRESENTATION USING GRAPH DATABASE

Yizhi Huang[1], Emmanuel Stefanakis[1*]

[1]Department of Geomatics Engineering, University of Calgary, Calgary-(yizhi.huang, emmanuel.stefanakis)@ucalgary.ca

KEY WORDS: Spatial Information, Geographic Information System, Multi-resolution, Multi-representation, Graph Database, Neo4j

ABSTRACT:

Multi-resolution representation has always been an important and popular data source for many research and applications, such as navigation, land cover, map generation, media event forecasting, etc. With one spatial object represented by distinct geometries at different resolutions, multi-resolution representation is high in complexity. Most of the current approaches for storing and retrieving multi-resolution representation are either complicated in structure, or time consuming in traversal and query. In addition, supports on direct navigation between different representations are still intricate in most of the paradigms, especially in topological map sets. To address this problem, we propose a novel approach for storing, querying, and extracting multi-resolution representation. The development of this approach is based on Neo4j, a graph database platform that is famous for its powerful query and advanced flexibility. Benefited from the intuitiveness of the proposed database structure, direct navigation between representations of one spatial object, and between groups of representations at adjacent resolutions are both available. On top of this, collaborating with the self-designed web-based interface, queries within the proposed approach truly embraced the concept of keyword search, which lower the barrier between novice users and complicate queries. In all, the proposed system demonstrates the potential of managing multi-resolution representation data through the graph database and could be a time-saver for related processes.

1. INTRODUCTION

A multi-resolution representation database is commonly defined as a structure that preserves linked representation, according to their degrees of geomatic or sematic abstraction, of the same spatial objects. (Sarjakoski, L. T. 2007). Therefore, how to well illustrate the links between the object, the abstraction level (or resolution), and the representation is the crucial part of constructing structures of such data. Existing approaches for spatial data show their limitation in many aspects. Tile-based structure, a method that is widely applied in web map services for its outstanding efficiency in information distributing, is rather time consuming when retrieving and updating representation(s) for spatial objects, since optimizing one spatial object may relate to reconstruction of all related tiles. Tree-based structures, which stores representation as independent entities, may be costly in traversal since queries within such structures may start from root nodes that are far from its target leaf nodes. The relational database, a mature technology, is displaying its limitation on fixed schema and expensiveness of join-style queries. To lower the cost of managing multi-resolution representation data, an alternative is needed. Neo4j, a No-SQL database platform that is high in flexibility and powerful in query, has proved its potential in fraud detection (Alsadi, Abuhamoud, 2018), recommendation engine (Stark et al. 2017) and social media analysis (Drakopoulos et al. 2017). In Neo4j, relationships between entities are explicitly and specifically defined, therefore with the query strategies of Neo4j, navigation between representation(s) could be simply performed within one query.

The main contribution of this paper is a comprehensive approach for storing, managing, and querying multi-resolution representation data on top of Neo4j. We construct an intuitive database structure that provides direct navigation and fast retrieving on representations as well as groups of representations. With this structure, storing and visualizing more types of geometries in Neo4j is available. In addition, we propose a concise and easy-operation interface that provides simplified functionalities for both novice and advanced users.

2. RELATED WORKS

2.1 Approaches for Storing, Querying and Managing Spatial Data

Existing approaches for operating spatial data mainly include tile-based methods, tree-based methods and methods based on relational databases. A tile in the tile-based method may refer to a pre-generated image, or a topological dataset that stores information of a certain area at a given resolution level. Google Map employs the “Tile Overlays” methodology for fast loading imagery and map data (MicroImages, 2012). In this pyramid structure, tiles are sorted according to the level of detail, and the higher the level, the more the tiles. For example, level 0 contains only one tile in the roughest detail, while level 1 would contain 2x2 gridded tiles with finer resolution. In CanVec Series, an official topological dataset of Canada, representations would be gathered into shapefiles according to their resolution (Natural Resources Canada, 2019). Therefore, retrieving representations at the same resolution could be a one-step process, while extracting representations for one multi-resolution spatial object may relate to duplicated search in every tile. This is mainly because connections between representations of the same spatial objects are not explicitly defined, but demonstrated through shared information, such as the name of the spatial object. Tree-based methods addressed this problem by chaining representations that refer to the same spatial object together, then organizing them according to their resolutions. In the Map Cube Model (Timpf, 1998), map elements are first divided into four categories, then their representations would be organized in decreasing levels of detail. However, as described in R-tree

* corresponding author

(Guttman, 1984), queries within such methods may start from the root node or parent nodes rather than hit the target directly. Approach in (Hampe M et al. 2004) optimized this problem by building links between one representation and all other one at lower resolution, which may increase the effort of updating. With more awareness of database control and operation, some approaches for spatial data are based on the existing database structure. The relational database is a mature technology that has been the fundamental of many desktop applications, e.g., the Geodatabase within ArcGIS (ArcGIS Desktop, 2017), in which one representation could be represented by multiple tables, and connections between each table is rather implicitly defined. The relational database has been proved to be suitable for well-structured data but has also claimed its limitation in join-style query (Van Bruggen, 2014). In addition, with rather static schema, relational database demonstrates its circumscription when facing the rapid growth of geospatial data. Research shows that Neo4j, a No-SQL database famous for its highly dynamic conception, could be more efficient in querying linking data (Stothers, J. A. et al. 2020, Gong, F. et al. 2018), which demonstrate the possibility of modelling multi-resolution representation.

2.2 Neo4j

Neo4j is a graph database designed and optimized for highly connected data (Miller, 2013). Elements in the graph database mainly include vertices, edges, and attributes. Vertices are considered as instances, and edges are the connections between them, while attributes are the further description of both vertices and edges. In Neo4j, vertices and edges are represented by nodes and single-directed relationships, and users are allowed to utilize those two components to create their own structure, or patterns in the world of Neo4j. Like nouns and verbs in sentences, the combination of nodes and relationships is truly intuitive in architecture. Attributes of nodes and edges are mainly represented by three indexable information units: property, label, and type, which all could be defined by users. The property holds detailed information for both nodes and relationships. Label and type are two identifiers that categorize nodes and relationships accordingly. Label in nodes is not strictly required and is unlimited in amount. However, the type for relationship is mandatory, and each relationship only belongs to one type. Figure 1(a) illustrates an example graph that demonstrates connections between three instances: Jack, Jennifer and the Neo4j company. Represented by two blue nodes, Jack and Jennifer share the label “person” and “developers”, and could be distinguished by their name, age, and location. In addition, both Jack and Jennifer work for the Neo4j company. Therefore, the two blue nodes are connected to the yellow one by relationships that are the same in type. Similar to this, the friendship between Jack and Jennifer is represented by two relationships that are the same in type, yet different in direction. This structure is truly intuitive since it is semantically akin to real-world expression. Note that querying on labels or types is not the only way to “grab” homogenous elements, when querying on properties that are not unique, multiple results are always expected.

Queries within Neo4j rely on Cypher, a declarative language that developed for highly connected data architecture. Queries in Cypher are performed linearly, which allows users to understand the whole process in an orderly way (Francis et al. 2018). Unlike another declarative language, SQL, Cypher prefers to express constraints with MATCH first, then use RETURN to have results at the end of each query. The number of results is decided by the number of constraints. Like painting a portrait of a person, the more the details, the clearer the appearance, and vice versa. Performing queries with no constraints would lead to output with all nodes or relationships in the database. Adding limited constraints would lead to return nodes or relationships at the same group, e.g.

MATCH (n1: developer) RETURN n1

MATCH ()-[r:Works_for]-() RETURN r

Distinguish queries within Neo4j mainly include pattern matching queries and path-finding queries (Van Bruggen, 2014). Pattern matching queries are like join style queries performed in relational databases, since they both find qualified instances through a series of constraints. Unlike relational databases, connections between instances are explicitly defined in Neo4j, join operations are transferred to hops from one node to another over those relationships.

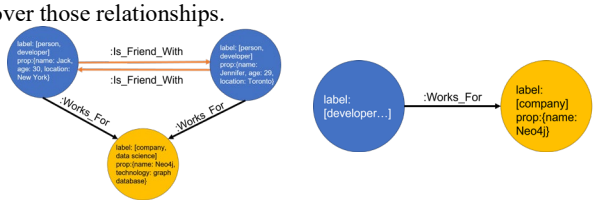


Figure 1. Example graph of Neo4j. (a)How a company and two developers that are friends to each other are connected. (b)Pattern defined within the example.

For example, when finding developers of Neo4j, the pattern is specified as Figure 1(b) shows:

MATCH(n1:developer)-
[:Works_For]->(n2 :company{name:"Neo4j"})
RETURN n1,n2

In such a pattern, the detailed properties are ignored since there is no constraint on, e.g., the name, age, or the relationships between developers. Note that in this process, only structures connected to starting node would be considered. Therefore, the query performance is rather independent of the size of data.

In relational databases, it may be the user who tells the database how tables are connected, while Neo4j is able to feed users the connections between instances through path-finding queries. In such queries, the user only needs to provide the starting and ending nodes, and Neo4j would find how they are connected. Take the structure in the figure above as an example, to find the connection between developers and Neo4j company, Cypher expression could be:

MATCH(n1:developer)-[r]-(n2:company{name:"Neo4j"})
RETURN r

Both pattern matching and path-finding queries demonstrate how Neo4j simplifies complex and expensive queries in relational databases, and eventually provides an efficient strategy for extracting adjacent instance(s). Although Neo4j has advantages in efficiency and flexibility in both dataset architecture and query, storing and visualizing spatial data within it is rather limited during our research period. In original Neo4j, point is the only geometry type that is supported in storing, querying and visualization. Hence, to store more types of geometry, Neo4j Spatial, a library that enables various spatial operations within Neo4j (Neo4j Spatial, 2010) is available. In addition, as an open-source platform, Neo4j provides all kinds of drivers, such as Java, JavaScript, Python, etc., which allow users to bridge between self-developed interface and database.

3. METHODOLOGY

3.1 Data and Resources

To perform thorough experiments on the proposed system, data applied in this research is expected to be rather diverse in

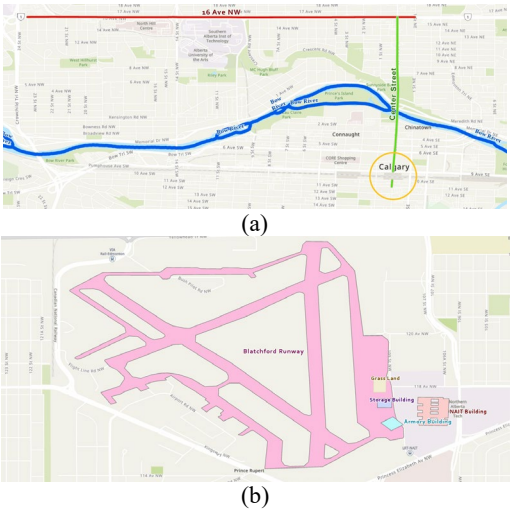
geometry types, category, and large in resolution range. Moreover, attention has also been paid to objects showing Intersects, Contains and Within, which are the most common spatial relationship queries in existing spatial databases. With those ideas in mind, CanVec series, an open-access dataset for the topographic data of the whole Canada, is selected as our data source. Spatial objects within are in eight themes, including Manmade, Hydrology, Transportations, etc. Resolution levels within CanVec series are available from 1:50k to 1:5M. Note that some manmade objects are only available at 1:50k resolution, while transportation objects are usually wider in resolution range. CanVec also feeds the demand for variety in both geometry types and spatial relationships. Said that, CanVec Series still has some fractures: 1) missing name or other identification, and 2) some representations at higher resolution are subdivided into multiple or even mislocated geometries. To address those problems, reference has been made to Google Map. Without changing the meaning of representations, some geometries are merged and repaired through QGIS (QGIS, 2021). However, for transportations represented by multiple LineStrings that are different in direction, the original representations are preserved, and direction-related properties are added when inserting into the proposed database. Overall, 34 diverse spatial objects are selected, and grouped into six maps according to their geolocation and spatial relationships, as illustrated in Table 1.

Map Name/ Map ID	Object	Resolution					Type	Theme
		1:15M	1:5M	1:1M	1:250k	1:50k		
City Center/ Map1	16 Ave NW			√	√	√	1	T
	Center Street			√		√	1	T
	Bow River			√		√	1	H
AB16HW/ Map2	AB16 Highway		√	√	√	√	1	T
Around 32 Ave/ Map3	32 Ave NE				√	√	1	T
	House1~5					√	2	M
	McCall Lake				√	√	2	H
BF Runway/ Map4	Armory Building					√	2	M
	Blatchford Runway					√	2	M
	Grass Land					√	2	M
	NAIT Building					√	2	M
	Storage Building					√	2	M
Lulu Island/ Map5	Building1~7					√	0	M
	Fraser River		√	√		√	1	H
	George Massey Tunnel		√	√	√	√	1	T
	Golf Course					√	2	M
	Golf Water					√	2	M
	Golf Wood					√	2	M
	Lulu Island		√	√		√	2	L
	Port Mann Bridge		√	√	√	√	1	T
	TriR Building					√	2	M
Around Lac St. Jean/ Map6	Lac Chigoubiche			√	√	√	2	H
	Lac St. Jean	√	√	√			2	H
	Railway	√	√	√			1	T
Type: 0 = Point, 1 = LineString, 2 = Polygon Theme: T = Transportation, H = Hydro, M = Manmade, L =Land								

Table 1. List of data

Since the WithinDistance query only performed from points, Building1~ 7 are specially included, and as manmade object they are at 1:50k only. Besides, unlike ideal multi-resolution database, resolution in CanVec may be inconsistent for some spatial objects, such as Bow River and Fraser River, which means the "next" resolution level may be different from prior knowledge.

Some fractured and concave objects are selected for verifying the geometry visualization, such as Center Street, Blatchford Runway and NAIT Building, as illustrated in Figure 2. Since Neo4j Spatial deploys WKT strings as one of the main geometry formats, representations are translated to WKT with the help of Get WKT for QGIS (Skeen, 2021) before constructing the proposed database. When developing the web-based interface, Turf.js (Turfjs, 2021), Terraformer (Terraformer, 2020) and Leaflet.js (Vladimir, 2019) are employed for more and better spatial queries, spatial information extraction, and geometry visualization.



(a)

(b)

Figure 2. (a)Fractured geometry Center Street at 1:50k resolution (circled), (b)Concave geometries: Blatchford Runway and NAIT Building.

3.2 System Structure

One of the ideal approaches for multi-resolution representation data is storing one representation for each spatial object, then generating other representations automatically. Although this approach could be a space-saver, it is limited by two factors. Firstly, spatial relationships between spatial objects should be consistent at any resolution, which may be an enormous challenge to the fully automated generation process. In addition, map generation is a time-consuming process and may influence the performance of real-time applications (Jones et al. 2000). Consequently, common approaches, for now, are still focusing on storing series of representations. A simple way is to store multiple interconnected representations, which, however, leads to further questions on 1) how to define the connections between representations, and 2) how to store properties for representations at different resolutions. When taking database performance and user experience as a priority, it would be better to define connections explicitly, and assign the whole set of properties into every representation (Spaccapietra et al. 2000). In existed methods, such as R-tree (Guttman, 1984), and (Hampe M et al. 2004), the specific IDs or address of the "next" representation are defined as attributes within representations for connection, such as Link _100k. Above the structure of data, (Jones, 1991) brought up awareness on the control of data updating, retrieving, and spatial process, which might be replaced by predefined data managing strategies deployed in well-developed databases, but still demonstrates the importance of cooperating database structure and query procedures.

To create a comprehensible and intuitive structure, we borrow the map-layer-feature concept from the traditional understanding of map sets, and semantically illustrate it on a tree-like structure.

As demonstrated in Figure 3, nodes for one multi-resolution representation structure include Map Node, Layer Node and Feature Node. The Map Node is the "leader" of a map and stores unique name and id of the map, while Layer Nodes are the "leader" of representations (of different spatial objects) at the same resolution. And layers are understood as groups of representations for different objects at the same resolution. Feature Node stands for representations. Information within Layer Nodes include the name and resolution value (referred to as "scale" in structure). Every Feature Node holds the whole set of information from the spatial object it represented, such as a name and directions (transportation object only), as well as a scale value and a geometry. For Layer Nodes that are included in the same map, they share the same layer name, and contain a scale value according to their resolution. Similar things happen to the Feature Nodes that represent the same spatial object: every node contains the same name in property and could be distinguished by scale value and other attributes, e.g., direction for Transportations. Storing the same information in different nodes may increase the information redundancy. However, this is the key for users to "grab" homogeneous elements in one query. For example, querying on one layer name would retrieve all Layer Nodes within one map structure, and querying on one feature name would retrieve all representations of the one spatial object, or detailed information of them. For easy-updating purpose, some information is gained through real-time query, e.g., spatial relationships. Storing spatial relationship within nodes may reduce the time for query, but every time a representation changed, all other representations must be updated synchronously.

Figure 3. System Structure.

Relationships between nodes are also defined in an intuitive way. Relationship Member_of and Feature_of are designed to chain nodes at different levels and allow fast retrieving on various levels of data. Between Layer Nodes and Feature Nodes stand for neighboring resolutions, Zoom_in and Zoom_out relationships are introduced, which allow users to navigate between representations, or layers, at adjacent resolutions. We did consider grouping those relationships with unique IDs or different names according to their destinations, but this would lose the advantage of retrieving representations at adjacent resolutions with limited information of the "next" resolution level. For example, relationships between 1:50k and 1:250k could be [:50_to_250], but for Bow River the 1:250k is absent. Therefore, [:50_to_1M] needs to be specially defined for the river, which increase both the storage size and the query difficulty. For now, all the traversal between adjacent resolution could be done with one query, since providing no resolution in both (n1) and (n2) would return all adjacent representations, and the more resolution information, the more specific the query result:

(n1)-[:Zoom_in]-> (n2) or (n1)-[:Zoom_out]-> (n2)

Although this structure is levelled semantically, queries could be performed from any nodes, or on any relationships within it. Take the map in Figure 3 as an example, to find which map that Lac St. Jean belongs to, the system just simply tracks through the Feature_of and Member_of relationship, then displays the id and/or name of the result Map Node. This design reduces the cost on queries and allows users to perform queries with limited knowledge of data.

To improve accessibility for users, and to provide visualization on geometries, a novel interface is also introduced, as illustrated in Figure 4. Framed with JavaScript, queries are simplified and re-grouped into more comprehensive and intuitive services based on their functionalities, including Find, Availability, Zoom, FindGeo, and GeoRelation. Keyword search strategy is thoroughly deployed when designing and simplifying those processes. Eventually, operations within functionalities above require no pre-earned knowledge of Neo4j, but only of the basic search process. In addition, Free Search, a functionality that requires full Cypher expressions and returns raw result streams, is also provided for advanced users. Results returned from queries are displayed in two parts, text area and map area, which would both be triggered in most of the cases. In the text area, properties of all kinds would be illustrated in a series of tables. Geometries are displayed upon map area, an interactive map that developed with Leaflet.js. Geometries in the result list are toggleable through a button displayed on the top of the text area. By zooming and dragging, users would gain some information outside of the database.

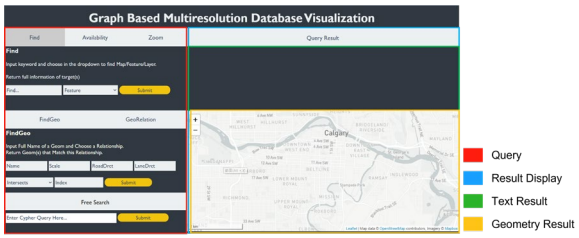


Figure 4. Layout of interface

4. FUNCTIONALITIES WITHIN THE PROPOSED DATABASE

4.1 Data Management

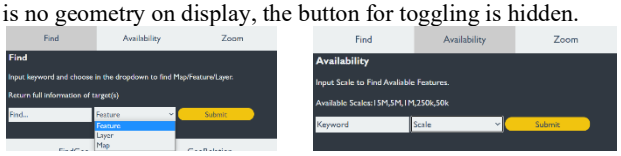
For now, data management is processed through Neo4j Browser (Neo4j, Inc, 2021), the original interface of Neo4j, which is mainly because Neo4j Browser provides a clear view of database structure with diverse approaches, which means database could be shown in node-relationships, or list of details. To avoid meaningless queries, in Neo4j Spatial, spatial queries are only appliable between geometries under the same layer. Therefore, Layer Nodes and Feature Nodes are introduced by inserting layers and then WKT geometries through Neo4j Spatial. However, the similar restriction is specially designed in other spatial operations to maintain consistency in spatial queries. There is no sequence between creating Map Nodes and the other two since there is no strict inheritance between nodes. Relationships are always created after nodes following the standard process of Neo4j, and all of them are created independently from nodes.

4.2 Global Search Function

In tile-based structures such as the CanVec Series, querying layers could be done in archives, while finding representations could only be done after opening layers in tools, e.g., ArcGIS. Here we group those queries into two functions: Find and Availability. They search for information of representations, layers, maps in the whole database, as illustrated in Figure 5. The Find function searches through the name properties. When searching at the feature level, users could provide a full name to

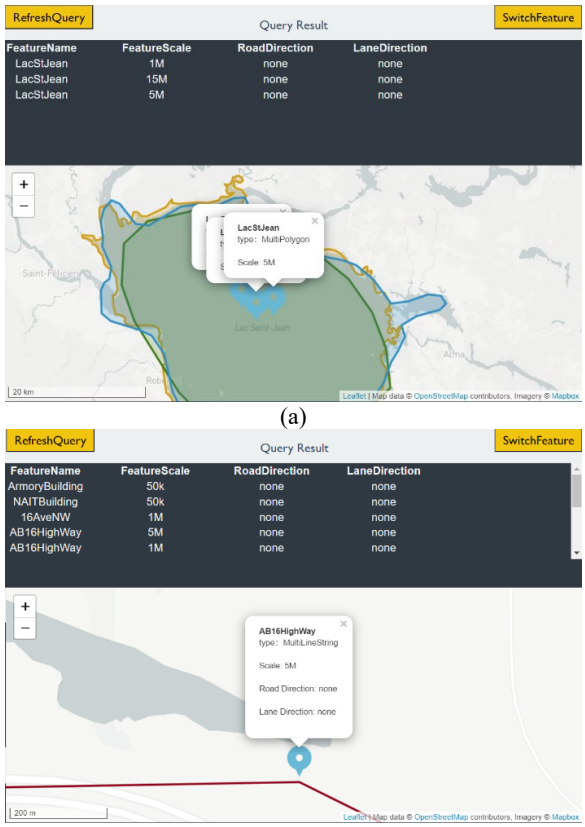
This contribution has been peer-reviewed. The double-blind peer-review was conducted on the basis of the full paper.
https://doi.org/10.5194/isprs-annals-V-4-2022-173-2022 | © Author(s) 2022. CC BY 4.0 License.

retrieve all the representations of one spatial object, as illustrated in Figure 6(a). When providing partial of name, the system would perform fuzzy queries and find representations that contain such field in name, see Figure 6(b). When finding layers and maps, both full-name and fuzzy quires would return information for both layers and maps in text result only, see Figure 7. Since there is no geometry on display, the button for toggling is hidden.



(a) (b)

Figure 5. Panel of functions. (a) Find (b) Availability.



(a)

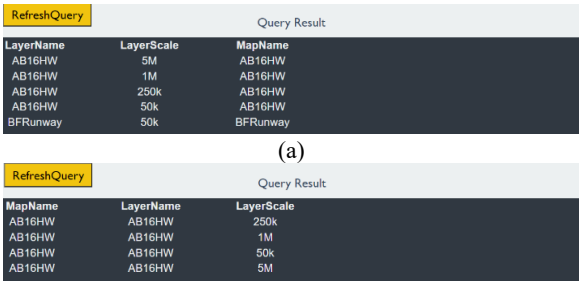
(b)

Figure 6. (a)Find all representations of Lac St. Jean. (b)A fuzzy query that finds representations containing "A" in name.

Results from both functions seem to be duplicated yet show the diversity of accesses to the proposed database. Since it is impossible for the developer to foresee the information preserved by users, we provide multiple functionalities that allow queries to be performed with even small pieces of information. The Availability function finds spatial objects that are available at the resolution inputted by users and extract all representations at a given resolution level within one click, see Figure 8. Both Find and Availability demonstrate how we balance between the information redundancy, and the fast information retrieving when designing the proposed system. For example, explicitly defining resolution value within every Feature Node allows performing the global search in a straightforward and efficient way. If the resolution value is inherited from layer or map level, the search process may involve multiple traversals.

4.3 Zoom Function

The Zoom function provides directed navigation between representations and layers at adjacent resolutions, as shown in Figure 9.



(a)

(b)

Figure 7. Results of finding (a) Layer that contain "B" in name. (b) Layer under the AB16HW map.

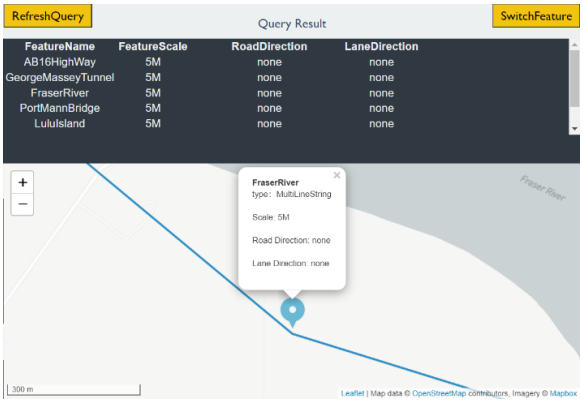


Figure 8. Results of finding spatial objects that are available at 1:5M resolutions.

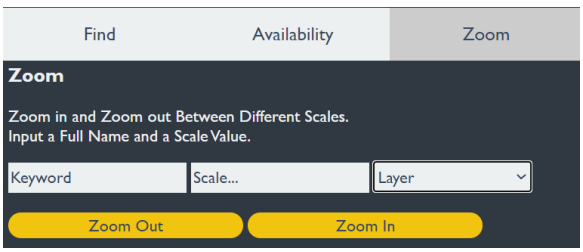


Figure 9. Buttons for the Zoom function are designed in a more straightforward way.

When querying for features, the result area would display both starting (inputted) and result representations, and by clicking the DuoSwitch button, both geometries would be rendered onto a base map simultaneously, see Figure 10(a). When rendering on more representations at either starting or destination resolutions, clicking the DuoSwitch button would toggle between them. The query for both layers and transportation object may return multiple representations. See Figure 10 (b) and (c).

The pattern matching query, which finds specific connections from a starting node, resemble the query strategy deployed in the Zoom function. In the Zoom function, the starting node(s) is defined through inputs, and the system finds result nodes that are connected to the starting one with [:Zoom_in] or [:Zoom_out] relationships. Therefore, different from Find and Availability, Zoom requires a full name and a scale value as input. Direct navigation between adjacent layers and representations mostly relies on the explicitly defined directed relationships between

This contribution has been peer-reviewed. The double-blind peer-review was conducted on the basis of the full paper.
https://doi.org/10.5194/isprs-annals-V-4-2022-173-2022 | © Author(s) 2022. CC BY 4.0 License.

both Layer Nodes and Feature Nodes. In tile-based maps where connections between representations are shown through shared names, jumping between layers at adjacent resolutions could be convenient, while jumping between representations for the same object may need to search in every related tile. In tree-based maps, jumping between representations may not be time consuming, yet there may not be a concept like layer in such structure, which limits its efficiency of traversal between them. In addition, in the proposed structure, since all adjacent representations or layers are connected through the same type of relationships, the unsure of the number of input and output will not affect query performance.

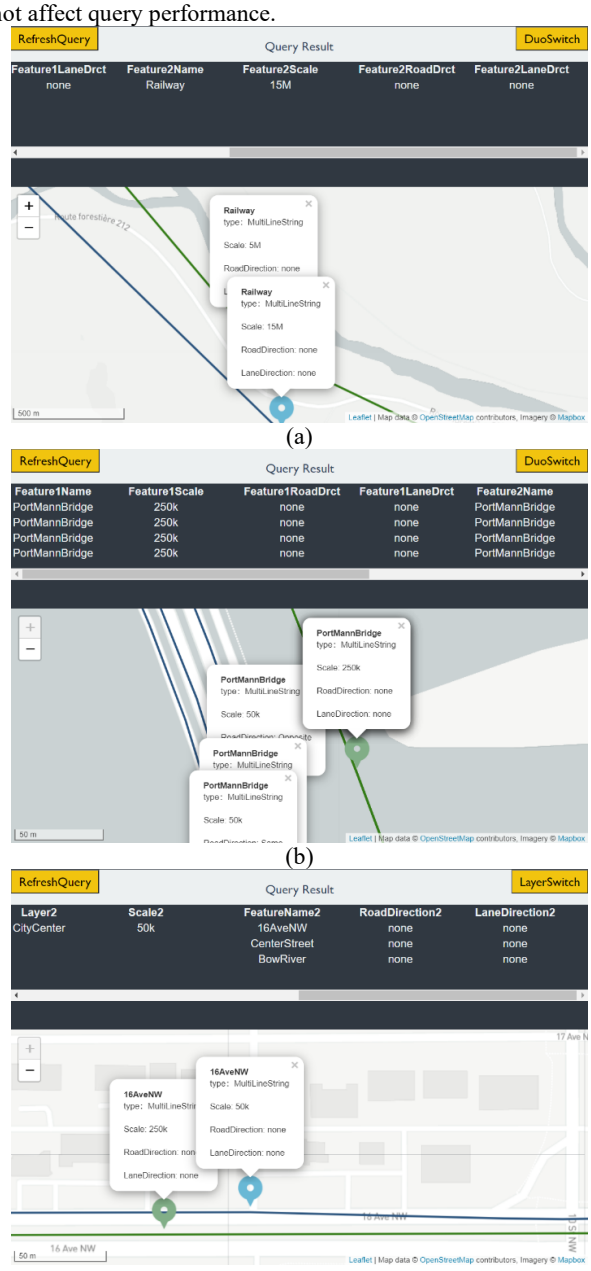


Figure 10. (a) Result of Zoom out from Railway at 1:5M. (b) Zoom in from Port Mann Bridge at 1:250k and receive an "uneven" result. (c) Navigating from the Layer Node CityCenter at 1:250k to 1:50k.

4.4 Spatial Query

Spatial queries in the proposed interface are performed through FindGeo and GeoRelation. The FindGeo function searches for object that is spatially related to the inputted one. The GeoRelation function takes two geometries as input, then returns the spatial relationship in between. In both functions, representations are specified through 4 factors: name, scale, road direction, and lane direction. Among them, the name and scale are mandatary, while the direction properties are only appliable to Transportations that have multiple representations. During one query process, only representations connected to the same Layer Node would be considered, since comparing representations at different resolutions is meaningless.

Spatial relationship types available in FindGeo include Intersects, WithinDistance, Contains, and Within. The first two queries are deployed with Neo4j Spatial, while the others are from Turf.js. The Intersects query finds representations that share any part of the inputted one, as displayed in Figure 11. WithinDistance searches for spatial objects that are within a certain radius from the given spot, which is appliable to point objects only. Unlike text results provided by other spatial queries, the calculated distance would also be displayed, see Figure 12. The Contains tells whether the second object is completely in the first one, while the Within shows the opposite results (Figure 13).

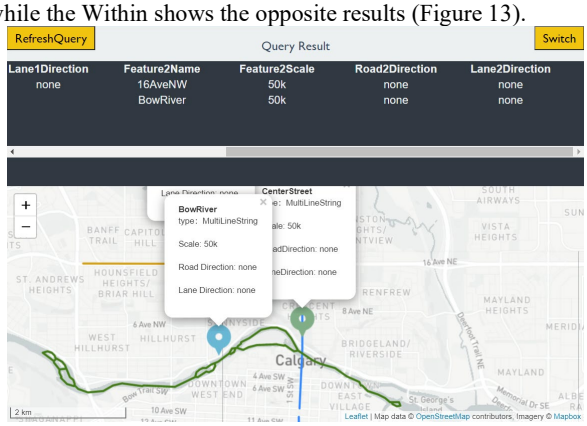


Figure 11. Results of geometries intersecting with CenterStreet at 1:50k.

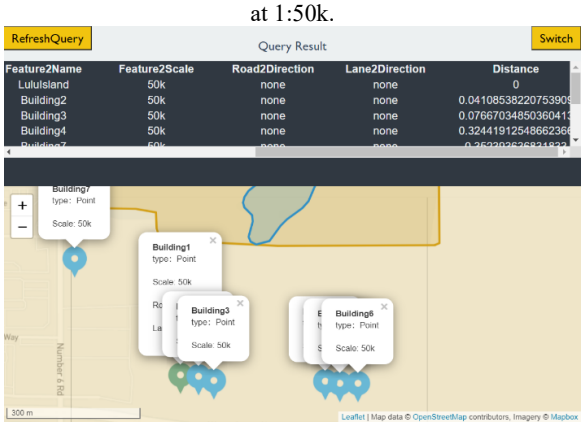


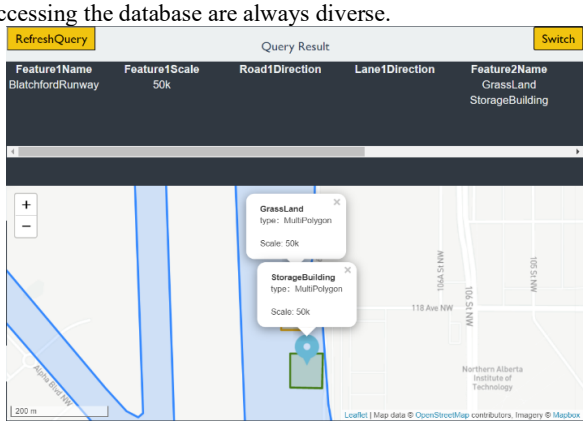
Figure 12. Result of finding objects that are within 0.5km from Building1. Since LuluIsland contains Building1, the distance in between is 0.

When having information of two representations yet their relationship is missing, users may consider performing several possible spatial queries. However, in the proposed interface, the GeoRelation is designed to address this problem in a straightforward way. In GeoRelation, the system would determine whether the inputted representations are Intersect,

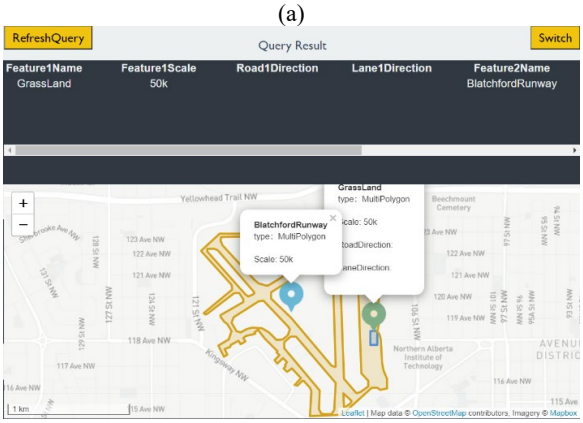
This contribution has been peer-reviewed. The double-blind peer-review was conducted on the basis of the full paper.
https://doi.org/10.5194/isprs-annals-V-4-2022-173-2022 | © Author(s) 2022. CC BY 4.0 License.

Contains, or Within, and displays the text result through boolean value, as illustrated in Figure 14. Queries in GeoRelation are supported by the boolean function within Turf.js. Note that point-point queries would be automatically detected and paused by the system since they are meaningless.

Both FindGeo and GeoRelation aggerates multiple spatial processes into one function, which reduces the time of jumping between different panels. Functionalities within FindGeo and GeoRelation may seem to be duplicated since they are both spatial queries. However, information held by users could be more limited than we expected, therefore approaches for accessing the database are always diverse.



(a)



(b)

Figure 13. (a)Results of finding geometries that are contained by Blatchford Runway. (b) Result of find geometries that Grass Land is within.

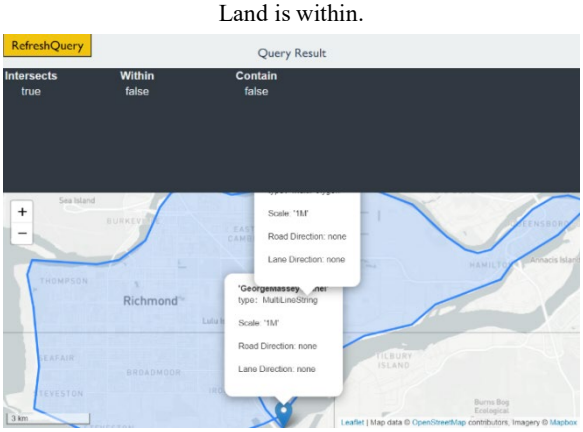


Figure 14. Spatial relationship between LuluIsland and George Massey Tunnel. Since they only intersect with each other, Contains and Within are displayed as "False".

4.5 Free Search

Functionalities in the sections above reduce the effort of learning and typing complicated queries. However, for undeployed queries, and for advanced users who prefer the original Cypher language, Free Search, a function that takes pure Cypher queries as input is developed. As its name, Free Search holds no limitation on the inputted queries, which means expressions here are unpredictable. Since properties that return from Neo4j must be claimed in queries before rendering onto the result area, results of Free Search would be illustrated as untranslated results, and visualization for geometries is unavailable, see Figure 15. Free Search is a function that not only designed out of frames, but also for gathering user-demanded processes in the upcoming development. It also provides a window for users to understand the transmission between Neo4j and the proposed interface.

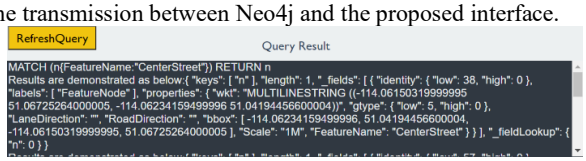


Figure 15. Result of Free Search

4.6 Error Proofing

Although processes deployed within the proposed interface is highly intuitive and accessible, error proofing methods are still needed to prevent users from performing meaningless queries and receiving confusing results. Methods deployed here are to initiative detect possible errors in the backend, and if misoperation is confirmed, the system would pause the undergoing session and warn the users with alerts. In the proposed system, we insert checkpoints for 1) invalid input. If the user submits the queries before providing enough information, the system will pop up an alert at the top of the website, which displays information accordingly. For example, "Please Input Value Before Submit!" goes for empty queries, while "Please Input Scale and Name Before Zoom In or Out" is for missing scale value only in Zoom. 2) No match for input. When submitting full information while input geometry(s) does not exist, the system would show "No match!" with some additional hint. 3) End of the result list. When clicking the toggle button and the display reaches the bottom of the result list, the system would popup with "End of Result! " to remind users. 4) Meaningless queries. WithinDistance function is designed for point only. When performing with other types of geometries, the system would inform users with "Please Input Point Feature". In addition, when performing spatial queries at different resolutions, the system would alert "Different Scale! Meaningless Spatial Query!". With a well-designed error proofing strategy, for now, the system thoroughly prevents users from retrieving meaningless query results, which increases the accuracy of the proposed functionalities. Meanwhile, hints and guidance provided through alerts not only pause current sections, but also help prevent misoperations in further processes. From this aspect, error proofing approaches could be considered as a tool that enhances system robustness.

5. CONCLUSION AND FUTURE WORK

In this paper, the graph-based system consists of two parts: an intuitive and understandable database structure, as well as a concise and accessible interface. With explicitly defined relationships between representations and layers at adjacent resolutions, direct navigation between them could be performed in a straightforward and easy-operation way. By sharing

information between nodes, we also achieve one-step data retrieving in both representation and layer levels. Benefited from the high tolerance of Neo4j, various geometry types and diverse spatial operations are now available by deploying multiple plug-ins. Cooperate with Neo4j Bolt Driver for JavaScript, more simplified functionalities and interactive geometry visualization are provided, which benefits both novice and advanced users, as well as the upcoming development. With the error proofing strategy, we secure the reliability of the proposed system. Overall, this paper demonstrates the potential of developing a comprehensive, efficient, and robust system for multi-resolution representation through Neo4j.

As for future work,1) More functionalities. For now, functionalities within are still basic. 2) Operations on historical queries. The proposed interface processes queries through individual sessions, which would be closed after displaying results. With operation for historical queries available, traversal between more resolutions will be possible. 3) Larger dataset and more users. Neo4j used in this paper is deployed in low computing power pc, which limits the number of users and the size of the dataset. Therefore, performance comparation with other approaches, e.g., relational database, are expected.4) Batch import. During our research, data importing process is accomplished step by step since the batch import is partially available in both original Neo4j and Neo4j Spatial.

REFERENCES

Alsadi, I.S. and Abuhamoud, N., 2018. Study to use neo4j to analysis and detection sim-box fraud. *Journal of Pure & Applied Sciences*, *17*(4), 31-35.

ArcGIS Desktop, 2017, Geodatabase system tables, desktop.arcgis.com/en/arcmap/10.3/manage-data/geodatabases/what-is-a-geodatabase.htm(21 June 2021).

Drakopoulos, G., Kanavos, A., Mylonas, P. and Sioutas, S., 2017. Defining and evaluating Twitter influence metrics: a higher-order approach in Neo4j. *Social Network Analysis and Mining*, *7*(1), 1-14. doi: 10.1007/s13278-017-0467-9

Francis, N., Green, A., Guagliardo, P., Libkin, L., Lindaaker, T., Marsault, V., Plantikow, S., Rydberg, M., Selmer, P. and Taylor, A., 2018. Cypher: An evolving query language for property graphs. In *Proceedings of the 2018 International Conference on Management of Data*,1433-1445.doi: 10.1145/3183713.3190657

Gong, F., Ma, Y., Gong, W., Li, X., Li, C., & Yuan, X.,2018. Neo4j graph database realizes efficient storage performance of oilfield ontology. *PloS one*, 13(11),e0207595.doi: 10.1371/journal.pone.0207595

Guttman, A., 1984. R-trees: A dynamic index structure for spatial searching. In *Proceedings of the 1984 ACM SIGMOD international conference on Management of data*, 47-57. doi: 10.1145/602259.602266

Hampe, M., Sester, M., & Harrie, L., 2004. Generating and using a multi-representation data-base (mrdb) for mobile applications. In *ICA Workshop on Generalisation and Multiple Representation, August,* 20-21.

Jones, C.B., 1991. Database architecture for multi-scale GIS. In *AUTOCARTO-CONFERENCE*- (Vol. 6, pp. 1-1). ASPRS AMERICAN SOCIETY FOR PHOTOGRAMMETRY AND REMOTE SENSING.

Jones, C.B., Abdelmoty, A.I., Lonergan, M.E., van der Poorten, P. and Zhou, S.,2000.Multi-scale spatial database design for online generalisation.In *9th international symposium on spatial data handling*, 34-44.

MicroImages, 2012. Google Maps Structure. www.microimages.com/documentation/TechGuides/78google MapsStruc.pdf (1 June. 2021).

Miller, J.J., 2013, March. Graph database applications and concepts with Neo4j. In *Proceedings of the southern association for information systems conference, Atlanta, GA, USA* (Vol. 2324, No. 36).

Natural Resources Canada, 2019. Topographic Data of Canada - CanVec Series-Open Government Portal. open.canada.ca/data/en/dataset/8ba2aa2a-7bb9-4448-b4d7-f164409fe056 (21 June 2021).

Neo4j,Inc, 2021, Neo4j Browser, neo4j.com/docs/browser-manual/current/(21 June 2021).

Neo4j Spatial, 2010, Neo4j Spatial, Version 0.26-neo4j-3.5, github.com/neo4j-contrib/spatial/tree/0.26-neo4j-3.5 (21 June 2021).

QGIS, 2021, QGIS, www.qgis.org/en/site/(21 June 2021).

Sarjakoski, L. T. 2007. Conceptual models of generalisation and multiple representation. *Generalisation of Geographic Information*,11-35.doi.org/10.1016/B978-008045374-3/50004-1

Skeen P,2021, QGIS3-getWKT,github.com/skeenp/QGIS3-getWKT, (21 June 2021)

Spaccapietra, S., Parent, C. and Vangenot, C., 2000, GIS databases: From multiscale to multirepresentation. In *International Symposium on Abstraction, Reformulation, and Approximation*, 57-70. Springer, Berlin, Heidelberg. doi: 10.1007/3-540-44914-0_4

Stark, B., Knahl, C., Aydin, M., Samarah, M. and Elish, K.O., 2017, September.Betterchoice:A migraine drug recommendation system based on neo4j.In *2017 2Nd IEEE international conference on computational intelligence and applications(ICCIA)*,382-386.doi:10.1109/CIAPP.2017.8167244

Stothers, Jessica A. M. and Andrew V. Nguyen., 2020, "Can Neo4j Replace PostgreSQL in Healthcare?"*AMIA Joint Summits on Translational Science proceedings.AMIA Joint Summits on Translational Science 2020,* 646-653.

Terraformer, 2020. Terraformer. terraformer-js.github.io/ (21 June 2021).

Timpf, S., 1998. Map Cube Model-a model for multi-scale data. In *8th International Symposium on Spatial Data Handling (SDH'98)*. Technical University of Vienna, Department for Geoinformation. doi: 10.3929/ethz-a-004364982

Turfjs, 2021, Turfjs/turf, github.com/Turfjs/turf/, (20 June 2021)

Van Bruggen, R., 2014: *Learning Neo4j*. Packt Publishing Ltd.

Vladimir, A., 2019, Leaflet, leafletjs.com/ (21 June 2021)