Using a Conceptual Model to Transform Road Networks from OpenStreetMap to a Graph Database

Dietrich Steinmetz[1], Daniel Dyballa[1], Hui Ma[2], and Sven Hartmann[1](✉)

[1] Clausthal University of Technology, Clausthal-Zellerfeld, Germany
{dietrich.steinmetz,daniel.dyballa,sven.hartmann}@tu-clausthal.de
[2] Victoria University of Wellington, Wellington, New Zealand
hui.ma@vuw.ac.nz

Abstract. We present a method for extracting road network data that has been crowdsourced in the OpenStreetMap project and transform it into a road network that is stored as a graph database. We propose an algorithm for the transformation, discuss opportunities for semantic enrichment of the road network, and for restricting the transformation to geographic regions of interest. Our approach is guided by a conceptual schema. To evaluate the practicability of our approach we have implemented it in a prototype tool, and conducted experiments that demonstrate the scalability.

Keywords: Geographic information · Data modelling
Graph database

1 Introduction

Graph databases represent data by graph structures with vertices, edges and properties. Recently a new class of DBMS has emerged that provide effective support for native storage, retrieval and management of relationship-centric data, such as Neo4j [14]. These systems are based on a simple property-graph model that permits efficient, highly scalable data traversals and avoids time-consuming joins. Meanwhile these systems also offer dedicated SQL-like query languages, such as Cypher in Neo4j. This makes them an attractive technology for applications that want to persistently store and process relationship-centric data.

In this paper, we focus on road networks that are relation-centric by nature. Road network data is an essential asset in many popular applications, such as route planning, navigation, ride sharing, traffic simulations, or urban planning. The database community has discovered road networks as a worthwhile study object, too [1,5,6,9–11,18–21,23,24]. While these works address a variety of different research problems, all of them conduct experiments to empirically explore the performance of proposed solutions. There is demand, both from industry and research, to make road network data readily available.

OpenStreetMap (OSM) [16] is a community project with the goal to generate a map of the world. The amount of road network data collected in OSM is comparable to commercial providers such as Google Maps. The advantage of OSM data is that it is free for personal and commercial use under the Open Data Commons Open Database License. OSM data can be downloaded in XML format. Unfortunately the way how OSM represents road network data is not relationship-centric. The conceptual data model of OSM can only represent geometric objects such as points, polylines or polygons. For example, road intersections and road segments are represented as points and polylines in OSM. However, the meta information that would allow the efficient traversal through these geometric objects is lacking. It is known that computing a route from a start point to an end point is not efficient when directly using OSM data [8].

We want to make road network data available in a graph DBMS so that it is directly accessible by applications that process such data. Our goal is to develop a method for extracting road network data from an OSM data set, and to transform it into a graph database which can be persistently stored and managed in a graph DBMS such as Neo4j. The input of the transformation will be an OSM data set in XML format. The elements of the conceptual data model of OSM will be mapped to the entities of a property-graph model.

We propose an algorithm for the transformation that defines when and how an element is transformed, and how its components are handled. Our main focus is on (1) single point objects that represent points of interest (POI) such as intersections, and (2) polylines that represent road segments, and determine course of the road. Obviously, not everything in an OSM data set is road network data. The input file has to be filtered to extract those elements that belong to the road network. Furthermore, it is important to control road types and the set of road properties that are considered during the transformation. The user should be able to select prior to the transformation which road properties are of interest for an application. While graph databases do not require a conceptual schema, we found it extremely useful to specify one to guide the transformation and show it is correct. It also provides valuable assistance in understanding the data, communicating requirements, and formulating semantic queries. The instance of the schema, i.e., the road network produced by the transformation, is stored in Neo4j where it is ready-to-use for applications, i.e., no further adaptation or conversion of the data is needed.

In many applications, it is essential to restrict the road network data to certain regions of interest. In order to support users in comfortably specifying regions of interest the course of borders such as district borders or national borders will be extracted from the OSM data and stored in the graph database, too. The stored border data will then be available for topological or analytical queries to be evaluated using a dedicated geographic library, such as Neo4j Spatial [15].

Organisation. This paper is organised as follows. In Sect. 2 we recall the conceptual data model of OSM. In Sect. 3 we give a formal definition of the problem and present the transformation for our proposed approach. In Sect. 4 we describe a prototypical implementation of our proposed approach. In Sect. 5

we report on experiments that we conducted to demonstrate the applicability of our proposed approach. In Sect. 6 we discuss related work. Finally, we conclude this paper in Sect. 7 and discuss suggestions for further research.

2 Background

The *conceptual data model of OSM* [16] provides three element type, see Fig. 1. *Node-elements* represent primitive objects located in a specific point on the map. A node-element may have attributes, such as *latitude* and *longitude* to specify the position on the map. *Way-elements* represent linear objects (called polylines) such as walkways, roads or boundaries. A polyline is a sequence of points. The points of a polyline will not be stored directly in the way-element, but rather references to the respective node-elements in an ordered list (called nd-list). If the last reference in the list is identical to the first one, then the polyline is actually a polygon. *Relation-elements* represent complex objects that often aggregate multiple node- and/or way-elements. A relation-element has a list of references to its member-elements (called member list). The attribute *type* of a member indicates whether it is a node- or a way-element. Each node-, way- and relation-element can have a set of *tags* associated to it. A tag provides a key-value pair and gives some property of the respective object.

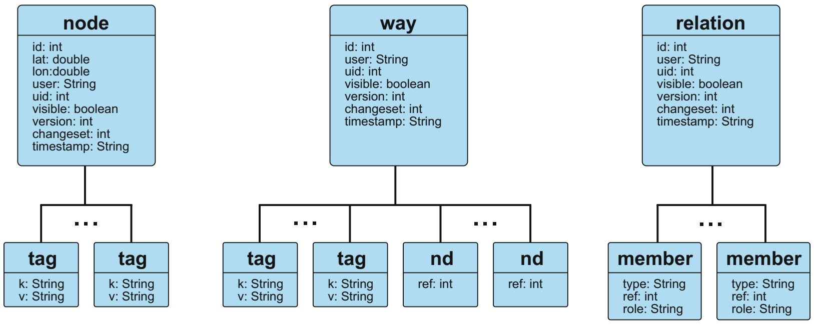

Fig. 1. Examples for the element types of the OSM data model.

Example 1. Figure 2 (left) shows a section of the OSM map for Bremen. In OSM, the road Frederikshavner Straße is represented by a set of polylines. The polyline that corresponds to the last road segment leading to the road intersection is marked in red.

The marked road segment is represented by the way-element with *id* “472359”, see Fig. 3. Its nd-list contains references to two node-elements. It has 13 associated tags, each of them carrying a key-value pair that specifies one road property, such as *surface* or *maxspeed*.

This polyline references two node-elements. See Fig. 2 (right) for its nd-list. Both node-elements have a set of attributes, including *latitude* and *longitude* that give their position on the map. The road intersection is represented by the node-element with *id* "349072703". It has an associated tag carrying the key-value pair (*highway*, "traffic_signals"). Hence, there are traffic signals are located in this point, which will be visualised accordingly on the map, see Fig. 2 (left).

OSM data can be exported to XML format (called OSM/XML in this paper) and downloaded from the website of the OSM project. For our example above, the respective XML is shown in Fig. 4.

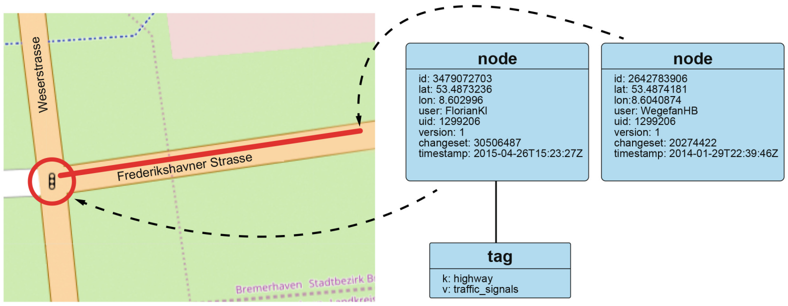

Fig. 2. A part of the road Frederikshavner Strasse. Two node-elements represent the ends of the marked road segment. The left-side one has traffic signals which are visualised accordingly. (Color figure online)

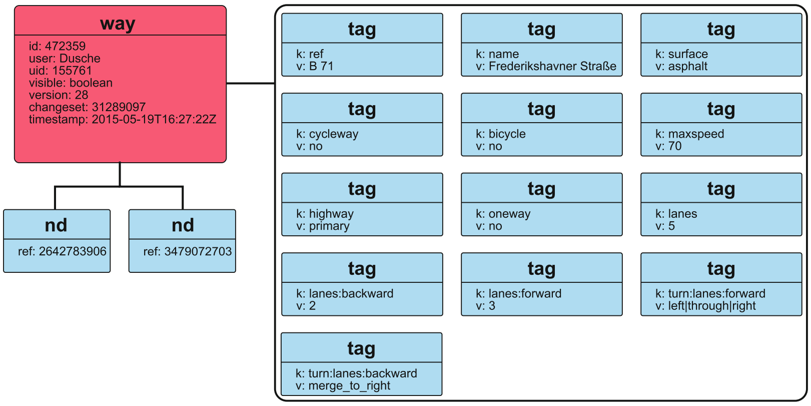

Fig. 3. The way-element that represents the marked road segment in Fig. 2.

```
<!--left node-element, traffic signal on intersection -->
<node id="3479072703" lat="53.4873236" lon="8.602996" ... >
    <tag k="highway" v="traffic_signals"/>
</node>
<!-- right node-element -->
<node id="2642783906" lat="53.4874181" lon="8.6040874" ... />
<!-- way-element -->
<way id="4723597">
    <!-- references to node-elements -->
        <nd ref="2642783906" />
        <nd ref="3479072703" />
    <!-- features and restrictions -->
    <tag k="name" v="Frederikshavner Strasse"/>
    <tag k="highway" v="primary"/>
    <tag k="surface" v="asphalt"/>
    ...
    <tag k="lanes:forward" v="3"/>
    <tag k="maxspeed" v="70"/>
</way>
```

Fig. 4. The OSM/XML fragment that corresponds to our example.

3 Method Conceptualization

In this section we will present our approach for extracting road network data from OSM and transforming them into a road network stored in a graph database.

3.1 Problem Definition

We introduce some terminology and formal notations. A *road network* is a directed graph $G = (V, E)$ where V is the set of vertices representing road nodes, and E is the set of edges representing road segments. Road nodes can be intersections, terminal points, bends, or similar. Each edge $e = (u, v) \in E$ connects two vertices $u, v \in V$.

Suppose we are given an *OSM data set* in the form of a triple $O = (N, W, R)$ where N is a set of node-elements, W is a set of way-elements, and R is a set of relation elements. Let $\mathcal{A}$ and K denote the sets of all attributes and keys, respectively. Every node-element $n \in N$ has a set of attributes $A_n \subseteq \mathcal{A}$ and a set of associated tags T_n, thus defining a partial function $tag_n : K \to dom$ such that $tag_n(k) \in dom(k)$ for all keys $k \in K$. Every way-element $w \in W$ has a set of attributes $A_w \subseteq \mathcal{A}$ and a set of associated tags T_w, thus defining a partial function $tag_w : K \to dom$ such that $tag_w(k) \in dom(k)$ for all keys $k \in K$. Moreover, it has a list $nd - list_w$ of references to nodes in N, and defines a partial function $next_w : N \to N$ such that $next_w(n)$ is the successor of n in the list $nd - list_w$.

Our goal is to transform an OSM data set into a road network and then store it in a graph database. The target of our transformation is a graph database that is based on the property-graph model, as for used e.g. in Neo4j [14]. We have developed the conceptual schema in Fig. 5 that will guide the transformation.

In our target schema, road nodes in V are captured by *node* vertices. They have at least the properties *latitude* and *longitude*. Further properties can be added if the user considers them relevant. The road type is captured by a label

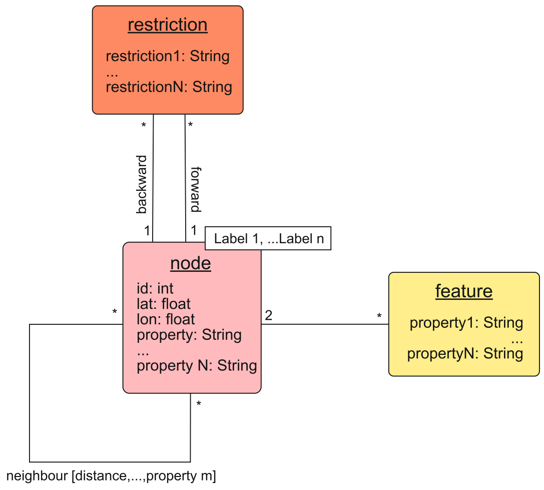

Fig. 5. Target schema of the transformation into a graph database

such as "motorway". In some applications it is important to distinguish between different road types. In taxi-ride sharing [6], for example, the pick-up and drop-off points of passengers cannot be on interstate highways. Labels can be easily accessed by taxi routing queries.

Road segments in E are captured by *neighbour* edges, each connecting two *node* vertices. They have at least the property *distance*. This enables shortest-path algorithms like Dijkstra or A^* to traverse the road network efficiently. Further properties can be added if the user considers them relevant.

Additional characteristics of road segments are captured by *restriction* and *feature* vertices. Experience shows that in Neo4j reading properties of an edge is much slower than reading properties of a vertex. To overcome this technical difficulty we propose a different solution: we simply add new vertices to the graph database to store the relevant properties of a road segment. In practical applications, it turned out to be advantageous to distinguish two kinds of tag keys: *features* that are independent of the direction in which the road is used, and *restrictions* that are dependent on the direction. For example, *surface* is a feature, while *maxspeed* is a restriction, see also Fig. 8.

3.2 Transformation of OSM Elements

To meet the requirements, we propose a transformation in two phases: first we focus on the type and course of the roads, and then we semantically enrich the road network. Our approach is outlined in Alg. 1. In principle, we map a node-element in the OSM data set O to a vertex in the road network G, and a way-element in O to one or more edges in G.

Note that not all information in an OSM data set is relevant for road networks. We are only interested in way-elements that represent road segments. These have an associated tag that carries a key-value pair (*highway*, road_type). The value herein specifies the road type. Possible values are "motorway", "trunk", "primary", "secondary", "tertiary", "unclassified", "residential", "service", etc. Further values are used to represent connections between roads of different types. For example, fedder roads that connect a normal highway to a motorway have the road type "motorway_link". We use the road type value to define a label for the vertices in G, see line 8 in Algorithm 1.

Example 2. Figure 6 shows two road segments of road type "motorway" as represented in OSM. Note that the two road segments are adjacent: both have a reference to the node-element with *id* "16" in their nd-list. The result of the transformation is shown in Fig. 7.

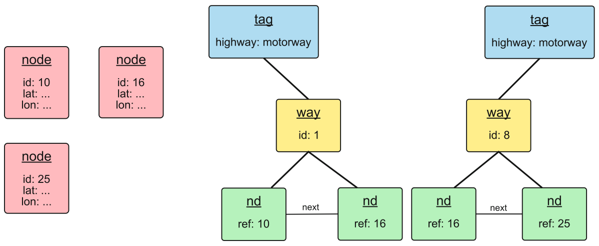

Fig. 6. Example of two way-elements that represent adjacent road segments

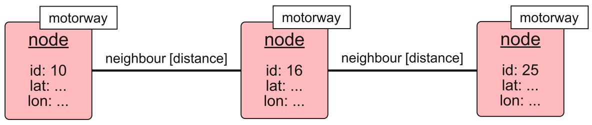

Fig. 7. The result of transforming the two way-elements in Fig. 6

The transformation of a node-element n is described in lines 5–15 in Algorithm 1. In this paper, we use the convention that only those node-elements will be mapped that belong to some way-element representing road segments. This ensures, e.g., that traffic signals will only be considered during the transformation if they are located on some road. Otherwise there could be traffic signals that are not related to any road, which is likely an error in the OSM data.

A node-element n may be referenced by multiple way-element, as in Fig. 6. We will map n to a single unique vertex in G, denoted by $image(n)$. In G one would need to search for a vertex the *id* of n. If no such vertex exists, a new vertex v must be generated in G. Afterwards the key-value pair of the linked tag will be copied to the new vertex.

Algorithm 1 Transformation

Input: An OSM/XML file representing an OSM data set $O = (N, W, R)$
Output: A Neo4j graph database representing a road network $G = (V, E)$

```
 1: for all way-elements $w \in W$ do
 2:     for all $ref \in nd - list_w$ do
 3:         let $n = id^{-1}(ref)$
 4:         if $image(n)$ is undefined then
 5:             create a new vertex $v$
 6:             put $id(v) := ref$
 7:             % define a label for $v$ based on the roadtype
 8:             add $tag_n(highway)$ as a label to $v$
 9:             copy every other user-selected attribute of $n$ as a property to $v$
10:             % check the semantic integrity of the tag set of $n$
11:             if $T_n$ is not valid then clean $T_n$
12:             % semantically enrich $v$
13:             copy every other user-selected tag of $n$ as a property to $v$
14:             insert $v$ into $V$
15:             put $image(n) := v$
16:         end if
17:         if $next_w^{-1}(n)$ is not undefined then
18:             let $u = image^{-1}(next_w^{-1}(n))$
19:             create a new edge $e = (u, v)$
20:             % define the type of the edge $e$
21:             put $type(e) := neighbour$
22:             copy every user-selected attribute of $w$ as a property to $e$
23:             insert $e$ into $E$
24:             % check the semantic integrity of the tag set of $w$
25:             if $T_w$ is not valid then clean $T_w$
26:             % semantically enrich $e$ with features
27:             create a new vertex $y$
28:             copy every valid user-selected feature tag of $w$ as a property to $y$
29:             create new edges $(u, y)$ and $(y, v)$
30:             insert $y$ into $V$ and $(u, y)$, $y, v$ to $E$
31:             % semantically enrich $e$ with restrictions
32:             create a new vertex $z$
33:             copy every valid user-selected feature tag of $w$ as a property to $z$
34:             create new edges $(u, z)$ and $(z, v)$
35:             put $type(u, z) := backward$ and $type(z, v) := forward$
36:             insert $z$ into $V$ and $(u, z)$, $(z, v)$ to $E$
37:         end if
38:     end for
39: end for
```

Note that not all attributes and tag keys stored in a node-elements will be of interest. It is up to the user to decide which ones she/he requires. We assume that the user selects those that she/he regards *relevant* prior to the transformation.

Then these ones will be copied during the transformation, see lines 9 and 13. In Fig. 7, e.g., the attributes *latitute* and *longitude* have been considered relevant.

The road course is determined by the nd-list of the way-element. It provides references to the node-elements that constitute the polyline, which are handled in lines 17–19. Often the tags associated with a way-element carry valuable information that should be kept. We copy relevant tags of w as properties to a new *feature* vertex and/or *restriction* vertex e, see lines 24–36. Note, however, that OSM itself does not provide such a classification of key tags. Hence, input from the user or a domain expert is needed.

Note that there are dependencies between the tags that have to respected for *semantic integrity*. We have indicated this in lines 11 and 25. In the tags associated with a node- or way-element, certain keys may only occur if others occur, too. For example, speed limits can only exist on road types. In Germany a speed limit is optional on motorways, while it is mandatory in New Zealand.

Example 3. To illustrate our approach, see Fig. 8. Tags that are relevant for the user can be stored in the feature node or in the restriction node. The results of the transformation are shown in Fig. 9.

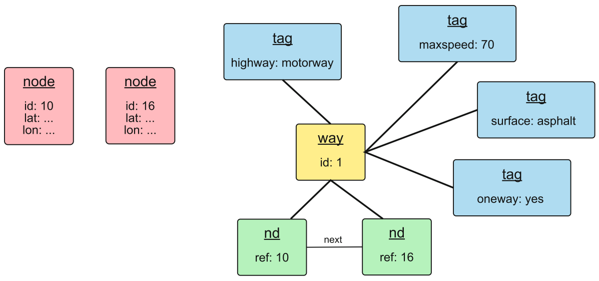

Fig. 8. An example of a way-element with tags carrying features and restrictions

For our transformation we can demonstrate that adjacency is preserved:

Proposition 1. *Two vertices u, v in G are adjacent (i.e., connected by a road segment $e = (u, v)$) if and only if there is a way-element w in O that has a tag with key* highway *and that references two node-elements m, n such that $image(m) = u$ and $image(n) = v$ and $next(m) = n$.*

3.3 Restriction to Regions

Finally, we extend our approach so that user queries against the road network can be restricted to specific geographic regions specified by the user. Actually, OSM provides a wealth of useful information on boundaries like national or regional borders. Neo4j Spatial [15] is a library of utilities for Neo4j that facilitates spatial

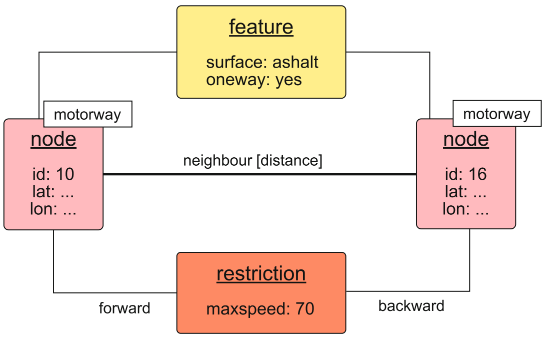

Fig. 9. The result of transforming the way-element in Fig. 8

operations on data. In particular, spatial indexes can be added to already located data, and spatial operations can be performed, e.g., searching for data within specified regions or within a specified distance of a point of interest.

To unlock these capabilities we have extended our conceptual data model to provide support for Neo4j Spatial. To achieve this we create a point layer in Spatial and add all vertices in our road network to this layer. Moreover, we create a WKT layer in Spatial to store and process polygons in WKT text format. The polygons that we add to the WKT layer correspond to boundaries on the map.

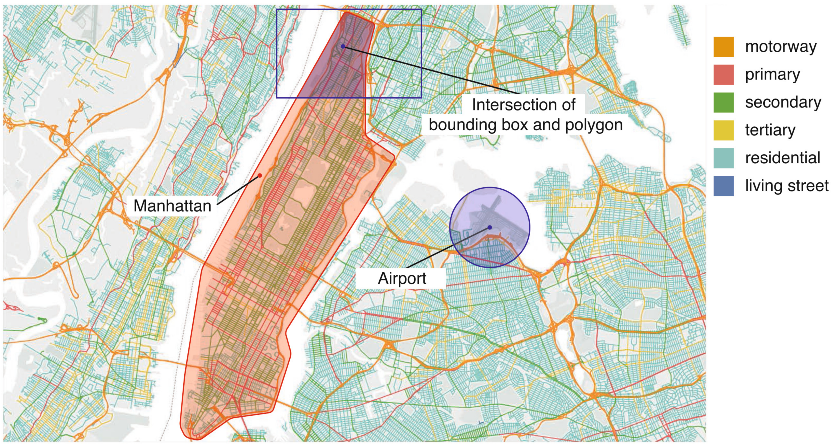

Fig. 10. Extension of our approach to support spatial queries

Therefore, we enhance our transformation to also extract and transform boundaries from OSM. They can be found using the key-value pair (boundary, “administrative”), and are usually represented by relation-elements in OSM. The points for the polygon are obtained from the member-list of the relation-element.

The polygons which we store in the WKT layer can be readily used as parameters for spatial queries against the road network in the graph database, see Fig. 10. Similarly, spatial queries that restrict query results to circles or bounding boxes are supported, too.

4 Proof of Concept

To demonstrate the practicability of our approach we have implemented a prototype system (called *OSM2RN transformer*) using Java. An OSM/XML file is used as input. It is imported, the OSM elements are filtered and transformed, and the results are stored into a graph database. For storing and managing the graph database we use the graph database management system Neo4j.

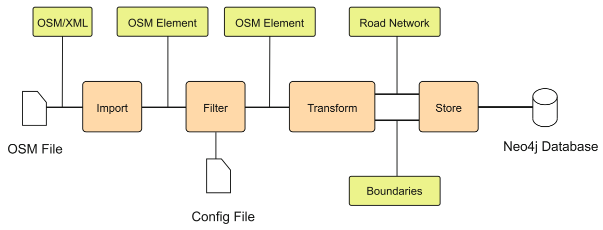

Fig. 11. Outline of the architecture of our OSM2RN transformer

The architecture of our tool, see Fig. 11, is based on the Pipes & Filter architectural pattern. It consists of four modules. The module *Import* takes an OSM/XML file as input and processes it using the SAX parser. Unfortunately, the ordering of the OSM elements in the OSM/XML file is not ideal for our purposes, as it starts with all node-elements followed by all way-elements followed by all relation-elements. For our transformation, the elements are needed in reverse order. Therefore we have to pass through the input file three times. The module *Filter* inspects each OSM element and its components whether they are needed, considering the general requirements and the user-specified rules. The module *Transform* executes the transformation. We obtain an instance of the target schema shown in Fig. 5. The module *Store* assembles the results of the transformation in Cypher operations and writes them into the Neo4j. When using Neo4j Spatial, then the extracted boundaries will be stored, too.

5 Case Study and Evaluation

In this section we will describe the test cases that we have used to evaluate our approach. In our experiments we have measured the execution time needed to transform an OSM data set and to store the results in the graph database. Every experiment has been repeated 30 times to eliminate measuring errors.

Testcases. We conducted experiments with two test cases to investigate the scalability of our approach. To explore the impact of using Neo4j Spatial, we conducted all our experiments with the library enabled and disabled. In *Testcase A*, we investigated how the size of the resulting road network impacts the run time. We used the OSM data set for Lower Saxony, see Fig. 12. The size of the resulting road network varied when different road types were considered during the transformation: in Test A1 we only considered motorway, motorway_link, in Test A2 also primary, primary_link, and in Test A3 also secondary, secondary_link.

	Testcase A: Lower Saxony	Testcase B: Berlin
#node-elements	26,897,284	4,622,951
#way-elements	31,676,139	4,957,448

Fig. 12. OSM data sets used for Testcases A and B

In *Testcase B*, we explored how the semantic enrichment impacts the run time. We kept the size of the road network constant, but varied the amount of features and restrictions that were considered during the transformation. We used an OSM data set for Berlin, see Fig. 12 and considered the following road types: motorway, motorway_link, primary, primary_link,secondary, secondary_link. In Test B1 we did not consider any features and restrictions, and then increased the amount stepwise in Tests B2 and B3 (in B1 features *tunnel, bridge, lanes, name* and restrictions *maxpseed*, in B2 additionally features *crossing, surface* and restrictions *motorcar, bicycle*).

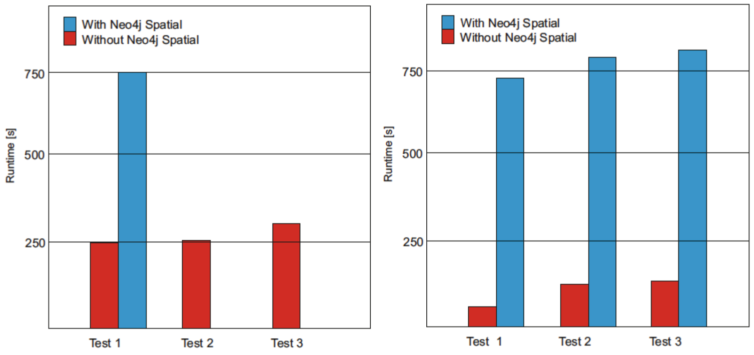

Fig. 13. Run times observed for Testcases A and B

Results. The run times are shown in Fig. 13, and the sizes of the resulting road networks in Fig. 14. In *Testcase A*, the size of the road network has at least doubled in each experiment. Without Neo4j Spatial, the run time increased from 248 s to 308 s for Test A1 to A3. When using Neo4j Spatial, Test A1 needed 748 s,

	Test A1	Test A2	Test A3
#road vertices	53,335	141,469	347,120
#road edges	53,896	143,519	353,214

	Test B1	Test B2	Test B3
#feature vertices	0	62,387	155,645
#restriction vertices	0	58,615	102,431

Fig. 14. Sizes of the resulting road networks for Testcases A and B

while Test A3 had to be aborted after 2 h. In *Testcase B*, the amount of semantic enrichment increased in each experiment. Without Neo4j Spatial, the run time increased from 58 s to 133 s for Test B1 to B3. When using Neo4j Spatial the run time increased from 731 s to 804 s for Test B1 to B3.

Our tests show that the use of Neo4j Spatial causes a performance loss. The insertion of geometry nodes into a layer in Spatial seems to be a time-consuming database operation. A possible explanation for this might be the build-in search index of the layer, which might require expensive reorganization after each insertion.

6 Related Work

The Neo4j Spatial library [15] also offers an importer for OSM data into Neo4j that can read OSM/XML files. In principle, it mimics the hierarchical organization of OSM/XML using node-elements, way-elements, and relation-elements. Different from our approach, it does not reflect the topology of the road network. Rather, the importer organizes the imported data such that it supports the spatial capabilities of Neo4j Spatial well. In particular, the importer does not generate a road network in form of a relationship-centric graph as defined in Sect. 3.1 which would simplify data traversal and speed-up computationally intensive applications such as navigation, ride sharing, traffic simulations or nearest neighbour queries. For that, further expensive transformations would be necessary to derive a suitable representation of the road network.

[22] studies the density and diversity of road networks in China based on OSM data for China, and finds that different parts of China observe different spatial patterns, and that road density reflects the intensity of traffic in the real world. This work could directly benefit from our tool as it unlocks the capabilities of Neo4j Spatial for analysing road network data and exploring spatial patterns.

[4, 17] present frameworks of methods for assessing OSM data quality solely based on the data's history. They propose quality indicators such as attribute completeness and average number of tags for points-of-interest (POI), or overall road length for road network completeness. When applying our approach to transform road networks into a graph database, such indicators can be efficiently evaluated by means of the data analytics capabilities of the Neo4j DBMS. Furthermore, the feature and restriction vertices that we have introduced in our conceptual database schema make the handling of historic data comfortable, as they can be versioned when data changes over time.

[3] explores the retrieval of geographic objects that are explicitly stored in OSM data, such as hospitals or lakes. [13] observes that there is a lack of methods for retrieving POI that are not explicitly stored, as this requires more detail

on geometries, topology and semantics. They propose rule-based spatial reasoning on OSM data using OWL and SQWRL, and discuss two applications for detecting entry points of footways and entrances of buildings. Similarly to our approach, it would be possible to design a conceptual model to extract the respective data for these applications from OSM, and store them in a graph database where they can be further analysed.

Conceptual modelling support for modelling and processing spatial data has been studied, e.g., in [7,12], but without focus on OSM, graph databases nor road networks. [2] proposes a rule-guided approach to resolve conceptual overlapping classes due to non-precise definition of geographic objects in OSM. [8] uses OSM data for personalized route planning, but without considering conceptual models nor graph databases. None of these works uses conceptual modelling to inform the transformation of road network data from OSM to graph databases.

7 Conclusions and Outlook

In this paper we have proposed a novel method for extracting road network data from OSM and transforming it into a road network in a graph database. To guide the transformation we have proposed a conceptual database schema for road networks that is based on the graph-property model and reflects the topology of the road network. We have discussed possible extentions and alternatives for the schema, described opportunities for semantic enrichment by user-selected relevant features and restrictions, and addressed the correctness of the transformation. We have further extended our approach by extracting and transforming boundaries from OSM that can then be used in the graph database to restrict user queries against the road network to geographic regions. We have implemented our approach in a prototype tool. Using this tool we have conducted experiments with several OSM data sets of different size. The results are very promising, and indicate that our approach is practical. Our tests have shown that our approach is more efficient without integrating the Neo4j Spatial library, but then the support for spatial queries is missing. It is up to the user to decide whether this extention is required for a particular application.

In future we plan to extend our approach to other information collected in the OSM project, such as railway network data.

References

1. Abeywickrama, T., Cheema, M.A., Taniar, D.: k-nearest neighbors on road networks: a journey in experimentation and in-memory implementation. PVLDB **9**(6), 492–503 (2016)
2. Ali, A.L., Sirilertworakul, N., Zipf, A., Mobasheri, A.: Guided classification system for conceptual overlapping classes in OSM. Int. J. GeoInf. **5**(6), 87 (2016)
3. Ballatore, A., Bertolotto, M., Wilson, D.C.: Geographic knowledge extraction and semantic similarity in OpenStreetMap. Knowl. Inf. Syst. **37**(1), 61–81 (2013)
4. Barron, C., Neis, P., Zipf, A.: A comprehensive framework for intrinsic OpenStreetMap quality analysis. Trans. GIS **18**(6), 877–895 (2014)

5. Chen, Z., Liu, Y., Wong, R.C., Xiong, J., Long, C.: Efficient algorithms for optimal location queries in road networks. In: ACM SIGMOD, pp. 123–134 (2014)
6. Cheng, P., Xin, H., Chen, L.: Utility-aware ridesharing on road networks. In: ACM SIGMOD, pp. 1197–1210 (2017)
7. Currim, F., Ram, S.: Modeling spatial and temporal set-based constraints during conceptual database design. Inf. Syst. Res. **23**(1), 109–128 (2012)
8. Graf, F., Kriegel, H.-P., Renz, M., Schubert, M.: MARiO: multi-attribute routing in open street map. In: Pfoser, D. (ed.) SSTD 2011. LNCS, vol. 6849, pp. 486–490. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-22922-0_36
9. Han, B., Liu, L., Omiecinski, E.: A systematic approach to clustering whole trajectories of mobile objects in road networks. IEEE TKDE **29**(5), 936–949 (2017)
10. Han, Y., Sun, W., Zheng, B.: COMPRESS: a comprehensive framework of trajectory compression in road networks. ACM ToDS **42**(2), 11:1–11:49 (2017)
11. Luo, S., Kao, B., Li, G., Hu, J., Cheng, R., Zheng, Y.: TOAIN: a throughput optimizing adaptive index for answering dynamic kNN queries on road networks. PVLDB **11**(5), 594–606 (2018)
12. Ma, H., Schewe, K.-D., Thalheim, B.: Geometrically enhanced conceptual modelling. In: Laender, A.H.F., Castano, S., Dayal, U., Casati, F., de Oliveira, J.P.M. (eds.) ER 2009. LNCS, vol. 5829, pp. 219–233. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-04840-1_18
13. Mobasheri, A.: A rule-based spatial reasoning approach for OpenStreetMap data quality enrichment; case study of routing and navigation. Sensors **17**(11), 2498 (2017)
14. Neo4j: Neo4j. https://neo4j.com/
15. Neo4j: Neo4j Spatial. http://neo4j-contrib.github.io/spatial/
16. OpenStreetMap: OpenStreetMap. https://wiki.openstreetmap.org/wiki
17. Singh Sehra, S., Singh, J., Singh Rai, H.: Assessing OpenStreetMap data using intrinsic quality indicators. Future Internet **9**(15), 1–22 (2017)
18. Steinmetz, D., Burmester, G., Hartmann, S.: A fast heuristic for finding near-optimal groups for vehicle platooning in road networks. In: Benslimane, D., Damiani, E., Grosky, W.I., Hameurlain, A., Sheth, A., Wagner, R.R. (eds.) DEXA 2017. LNCS, vol. 10439, pp. 395–405. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-64471-4_32
19. Wang, S., Xiao, X., Yang, Y., Lin, W.: Effective indexing for approximate constrained shortest path queries on large road networks. PVLDB **10**(2), 61–72 (2016)
20. Yan, D., Zhao, Z., Ng, W.: Efficient algorithms for finding optimal meeting point on road networks. PVLDB **4**(11), 968–979 (2011)
21. Zhang, D., Yang, D., Wang, Y., Tan, K., Cao, J., Shen, H.T.: Distributed shortest path query processing on dynamic road networks. VLDB J. **26**(3), 399–419 (2017)
22. Zhang, Y., Li, X., Wang, A., Bao, T., Tian, S.: Density and diversity of OpenStreetMap road networks in China. J. Urban Manag. **4**(2), 135–146 (2015)
23. Zhao, J., Gao, Y., Chen, G., Jensen, C.S., Chen, R., Cai, D.: Reverse top-k geo-social keyword queries in road networks. In: IEEE ICDE, pp. 387–398 (2017)
24. Zheng, B., Su, H., Hua, W., Zheng, K., Zhou, X., Li, G.: Efficient clue-based route search on road networks. IEEE TKDE **29**(9), 1846–1859 (2017)