Modelling and building of a graph database of multi-source landmarks to help emergency mountain rescuers

Véronique Gendner[a], Marie-Dominique Van Damme[a], Ana-Maria Olteanu-Raimond[a]

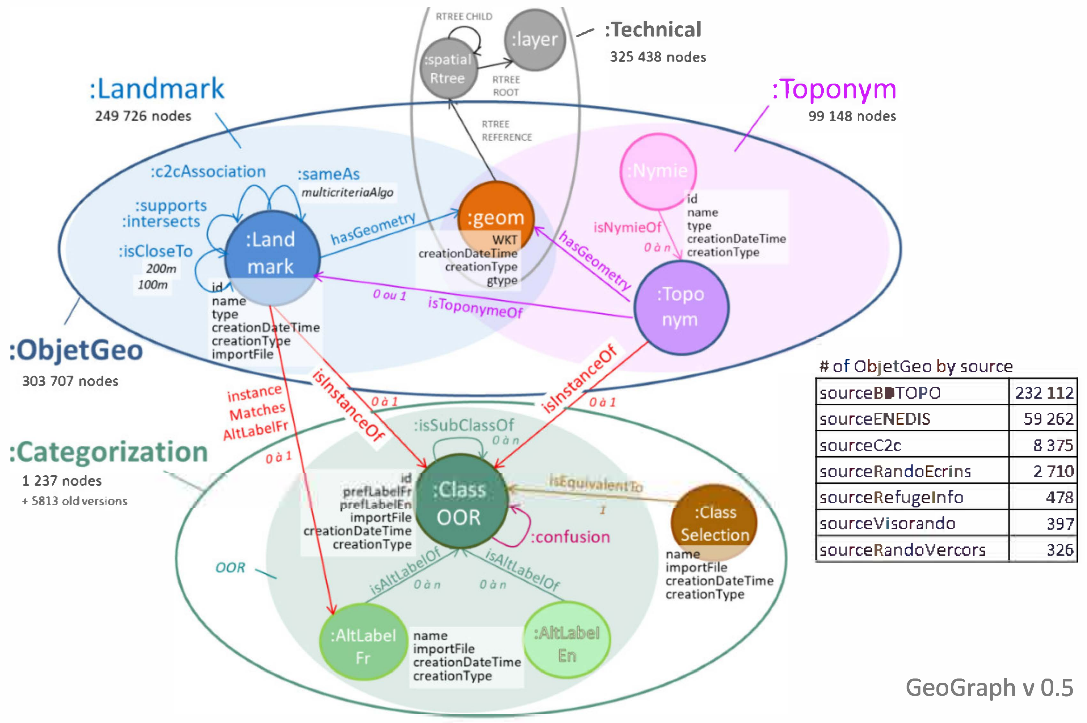
[a] *LASTIG, Univ Gustave Eiffel, ENSG, IGN, F-94160 Saint-Mande, France,*
veronique.gendner@ign.fr; marie-dominique.van-damme@ign.fr; ana-maria.raimond@ign.fr

Keywords: graph database, modelling, multi-source spatial data, data analysis, normalized categorisation

The purpose of the Choucas research project[1] is to come up with methods, tools and resources, to help mountain rescue team localise victims, when answering emergency calls. In this context, we have built a graph data base with the Neo4j[2] Labelled Property Graph (LPG) technology, that integrates several sources of geolocated objects. Some data comes from the national mapping agency (IGN, BDTOPO), others, like routes from crowdsourcing websites (see table on Figure 1). Imported data has been categorised based on the *Landmarks Objects Ontology (OOR)*[3] that had previously been produced by the project team. The flexibility of graph databases helps make the right modelling choices by progressively taking into account problems observed in the data as well as researchers and users feedback and new needs.

Iterative Modelling

With Labelled Property Graph data bases, data is modelled with nodes and relations that can both have properties. Relations have one and only one type. Nodes can have multiple labels that organises them in a set-theoretic way. Taking advantage of the flexibility of graph databases that allow for technically easy modifications of initial modelling choices, an iterative process of data analysis leading to a model hypothesis, data analysis with the model leading to model adjustments etc. has led to the schema of nodes and relationships presented in Figure 1.



# of ObjetGeo by source

sourceBDTOPO	232 112
sourceENEDIS	59 262
sourceC2c	8 375
sourceRandoEcrins	2 710
sourceRefugeInfo	478
sourceVisorando	397
sourceRandoVercors	326

Figure 1. The proposed Graph DB is named GeoGraph. It's simplified schema has three main types of nodes: geographical objects (`:ObjetGeo`), node used for objects categorisation (`:Categorization`) and technical nodes (`:Technical`).

Nodes representing geographical objects (i.e. landmarks and toponyms – these name choices still being worked on) have a minimum set of properties: `id` and `name` from the source, as well as a `type` that normalises the name of source attributes containing categorization information. `creationType` and `importFile` trace information about the origin of the node (*import, DB query, spatial intersect,…*). The link between homologous objects coming from different sources are modelled by the *sameAs* relationships. A property of the relation, specifies that is has been computed with a multicriteria matching algorithm[4].

Relationships between geographical objects (e.g. *hasGeometry, isToponymeOf, isInstanceOf, sameAS, closeTo…*) are either imported or calculated. For some spatial relationships, the Neo4j spatial plugin[5], based on the JTS library has been used.

Toponyms and corresponding landmarks that are represented in different tables in the source data base IGN BDTOPO are modelled as a subgraph (see figure 2). It represents the fact that toponyms can have different names. This allows for easy search on the different identified names or spelling for a same toponym (`Nymie`) and its possibly related landmark.

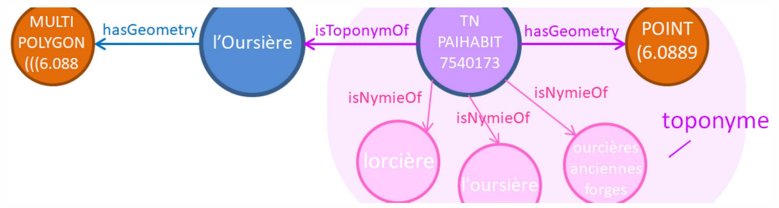


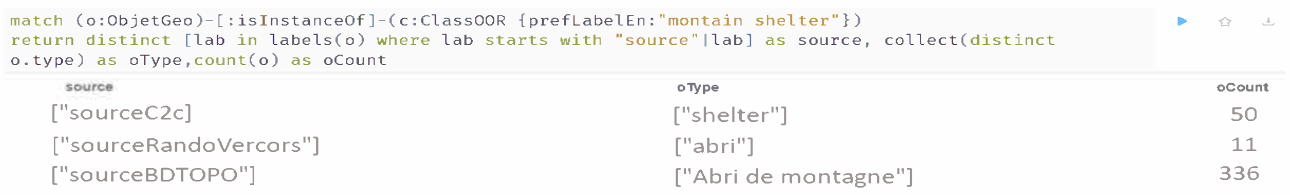
Figure 2. Example of a toponym modelled as a subgraph : a `Toponym` node, three `Nymie` (*lorcière*, *l'oursière*, and *ourcières anciennes forges*) and its geometry. In this way, relationships that, in the source, are only expressed by identical ids are now explicit and very efficient to traverse, by essence of graph DB architecture.

Data categorisation and Ontology instantiation

The *Landmarks Objects Ontology (OOR)* has been imported in the DB and a set of rules using the `type` and `name` properties of nodes has been defined to link objects to the corresponding class, thus normalising the different categorisation classes that are provided by the different sources and also instantiating the ontology with imported data.

Data analysis

The flexible schema has also shown very useful during the building process of the database, to explore, analyse and compare data, regardless of different versions of the data and the source it comes from.



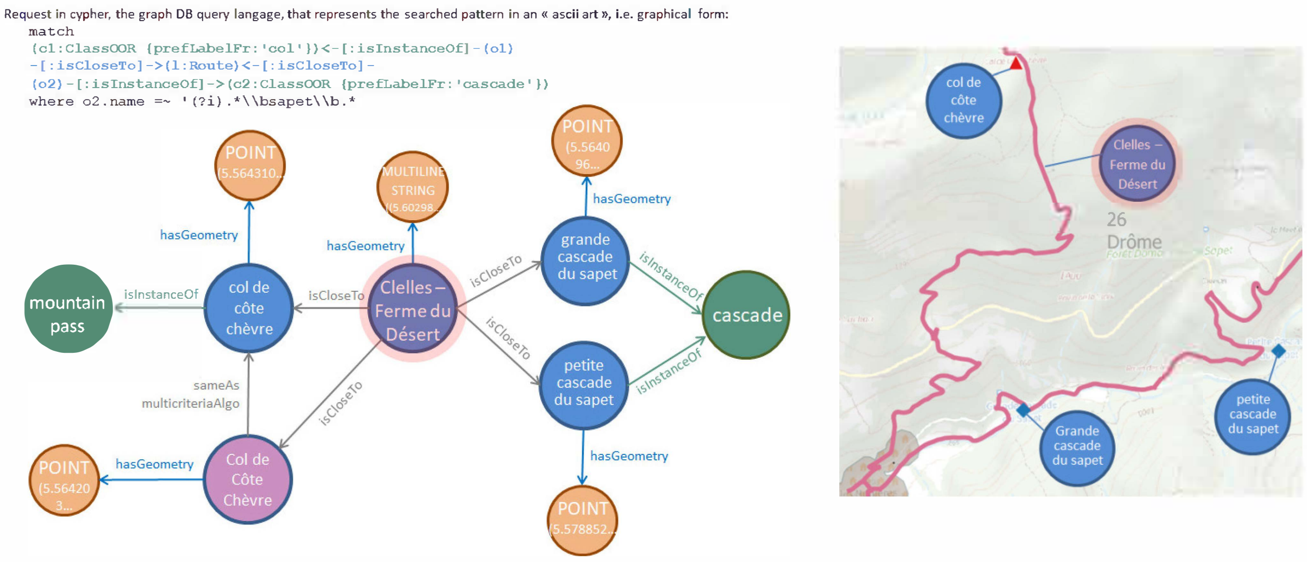
```
match (o:ObjetGeo)-[:isInstanceOf]-(c:ClassOOR {prefLabelEn:"montain shelter"})
return distinct [lab in labels(o) where lab starts with "source"|lab] as source, collect(distinct
o.type) as oType,count(o) as oCount
```

source	oType	oCount
["sourceC2c]	["shelter"]	50
["sourceRandoVercors"]	["abri"]	11
["sourceBDTOPO"]	["Abri de montagne"]	336

Figure 3. Query returning the number of objects in each source, with the different `type` value found in the source.
(Instantiation rules creating `isInstanceOf` relationships have previously been run to connect objects to the corresponding class.)

Searching objects by pattern matching on the graph

Searching a graph DB can be done either on nodes and relationships property strings (e.g. `node.name="sapet"`), but also by matching a pattern in the graph based on the labels of nodes and their relationships (see Figure 4).



Request in cypher, the graph DB query langage, that represents the searched pattern in an « ascii art », i.e. graphical form:

```
match
(c1:ClassOOR {prefLabelFr:'col'})<-[:isInstanceOf]-(o1)
-[:isCloseTo]->(l:Route)<-[:isCloseTo]-
(o2)-[:isInstanceOf]->(c2:ClassOOR {prefLabelFr:'cascade'})
where o2.name =~ '(?i).*\\bsapet\\b.*
```

Figure 4. If the victim says "*I'm on the route to the Sapet cascade, I've passed a mountain pass*", querying the DB on the graph pattern shown left, allows to look for a route that passes both a cascade that contains "*sapet*" in its name and a mountain pass. The result is the route called *Clelles - Ferme du Désert.*

Perspectives

A first perspective of this work includes further adjusting the ontology and instantiation rules by further analysing how the *OOR* is instantiated with current data, as well as adding new sources of data and validating the instantiation by involving the users (mountain rescue). Another perspective is to formally define and calculate spatial relations between geographical objects (*isCloseTo*, *see*, *crosses*, etc.) in order to add them to the DB. Representing relief, for example, to be able to detect if close objects are actually separated by a cliff will also be considered. The spatial relations will instantiate different types of spatial relations defined by the ontology of spatial relations already proposed in the Choucas project team[6] and this ontology will also be enriched with new spatial relations. The multicriteria matching algorithm could be implemented as a Neo4j plugin, to easily identify newly imported nodes representing the same real world entity. Finally, further work on how to formally define a route will be done based on work also already initiated in the Choucas project[7].

[1] http://choucas.ign.fr/ [2] http://www.e-tissage.net/neo4j-for-newbies/ [3] http://choucas.ign.fr/doc/ontologies/index-fr.html
[4] https://doi.org/10.1080/23729333.2019.1615730 [5] https://neo4j-contrib.github.io/spatial/
[6] https://hal.archives-ouvertes.fr/hal-02187587 [7] https://doi.org/10.1080/23729333.2019.1615730