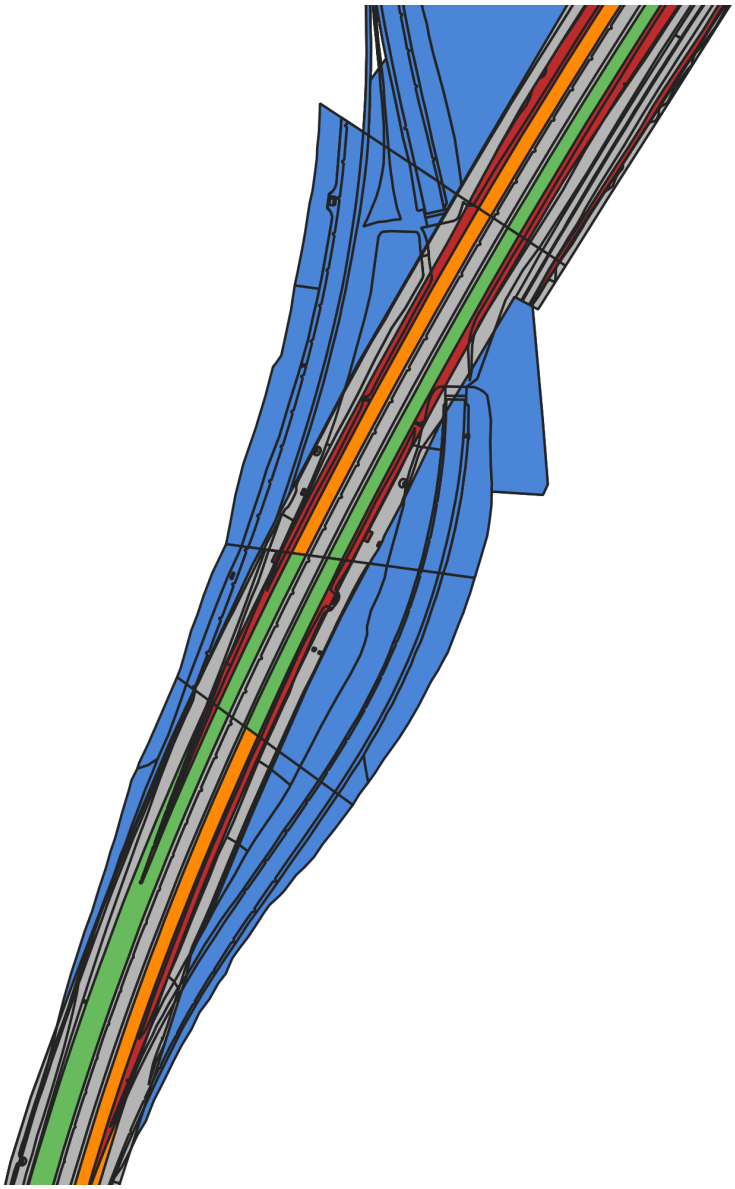
Synchronization of two related geospatial databases using dates

Karin Jacquelien Staring
student number #4952510
karinstaring@msn.com



June 26, 2020

Internship Report Geomatics for the Built Environment

Karin Staring: *Synchronization of two related geospatial databases using dates* (2020)

This report forms the written record of an internship carried out in the period 21 april - 21 june 2020. The internship is an elective of the MSc Geomatics for the Built Environment, Delft University of Technology, awarded with 5 ECT.

Geomatics tutor	Dr. Martijn Meijers
Company	Rijkswaterstaat
Internship supervisor	Dr. Tessa Eikelboom

Abstract

Rijkswaterstaat (RWS) has to maintain two related geospatial databases both with their own data model. The Basisregistratie Grootschalige Topografie (BGT) database contains all physical objects in the Netherlands, which is maintained by different data owners [RWS, 2020a]. RWS is the data owner for all national roads and national waterways. The data, that is used for the BGT in the RWS data owner area, is mainly based on the internal Digitaal Topografisch Bestand (DTB) database of RWS. The two geospatial databases became asynchronous, which indicates that the updates have not been done in both databases. One could argue that maintaining two large geospatial databases, that both receive updates , is impossible manually. Although a two-sided data mapping would be optimal for databases, that both receive updates, an one-sided mapping, an excel file, from the DTB to the BGT were used. The accuracy of the objects could be detected from interpreting aerial images or the dates. The reason that the dates attached to the objects were used is because interpreting aerial images would require more advanced algorithms.

The scope of the research is limited to the available one-sided data mapping, polygons and the ground layer in both databases to avoid overlapping polygons in the individual databases. The problem statement, together with the scope of the research resulted in the following main research question: Could dates be used as indicator for the synchronization of two related geospatial databases containing polygons?

A general method were established to answer the main research question. Firstly, the DTB objects were mapped to the BGT objects in order to make a comparison with the BGT. A unique row in the excel file were selected for this. The row contains DTB attributes, which were used to select the row, and the corresponding BGT attributes, which were added to the DTB object. In order to make the comparison, the converted DTB and BGT had to be referenced to each other. An advantage of the union is that the attributes could be compared for every made geometry, where the databases overlap. In this way, the geometries could not be compared anymore, but it became visible, where the attributes differ. However, it requires further processing to synchronize the objects. Fortunately, the geometries include the `bgt_id` and `dtb_id` of the original objects, which would allow us to synchronize the outdated objects afterwards, but this has not been done. After the union, their attributes have been compared to determine, which objects differ from each other. If they differ, it were investigated, which object is more up-to-date by comparing the dates. The comparison checks both if it is a different object and if it is a different mutations. This information could be used for synchronizing the databases.

Unfortunately, the main research question could not be answered, because of these reasons:

1. The excel file most likely differs from the conversion, that were used to update the BGT. As a consequence, most objects could not be mapped to a row in the excel file or they were mapped to a different row with different attributes. For this reason, they differ from the corresponding BGT object.
2. Synchronizing the two related geospatial databases requires further processing as stated earlier. The results were also not compared with the ground truth, which must be done to determine the accuracy of the synchronization.
3. Most BGT objects seemed more up-to-date than the DTB objects. This is unlikely, because DTB objects are used to establish the BGT in the RWS data owner area. This indicates that the threshold of 180 days is probably too low or that it is caused by the first delivery in 2017.
4. The same DTB object could correspond to multiple BGT objects at the same location. The reason for this is that the BGT contains multiple overlapping polygons at the ground level. This results in multiple references at the same location, which results in different comparisons at the same location, which makes the outcome of the comparison ambiguous.

Contents

1. Introduction **1**
- 1.1. description of the organisation and the department . . . 1
- 1.2. background . . . 2
- 1.3. description of the project . . . 2

2. Method **4**
- 2.1. select area: BGT and converted DTB . . . 4
- 2.2. referencing the BGT and the converted DTB . . . 6
- 2.3. comparison between the BGT and the converted DTB . . . 7
  - 2.3.1. comparison between the dates . . . 7

3. Data and software **10**
- 3.1. data . . . 10
- 3.2. software . . . 11

4. Implementation **13**
- 4.1. select area: BGT and converted DTB . . . 13
  - 4.1.1. DTB . . . 13
  - 4.1.2. BGT . . . 14
- 4.2. referencing the BGT and the converted DTB . . . 15
- 4.3. comparison between the BGT and the converted DTB . . . 16
  - 4.3.1. comparison between the dates . . . 18

5. Results **19**
- 5.1. incorrect DTB data mapping . . . 19
- 5.2. 180 days threshold seems too low . . . 19
- 5.3. multiple overlapping BGT objects at the ground level . . . 19

6. Conclusion **24**

7. Recommendation **26**

8. Reflection **28**

A. Appendix **29**
- A.1. MMS notification . . . 29
- A.2. DTB quality . . . 30
- A.3. database creation . . . 31
- A.4. data mapping function . . . 32
- A.5. delete and rename DTB attributes . . . 33
- A.6. The function `union_geometries` . . . 34
- A.7. The function `compare_attributes` . . . 35
- A.8. The function `compare_dates` . . . 36

Acronyms

RWS Rijkswaterstaat 4
IenW Ministerie van Infrastructuur en Waterstaat 1
DTB Digitaal Topografisch Bestand 4
BGT Basisregistratie Grootschalige Topografie 4
MMS Mutation Reporting System 2
IMGEO Informatiemodel geografie 2
LV Landelijke voorziening 2
ID Identifier 6
PDOK Publieke Dienstverlening Op de Kaart 10
GIS Geographic Information System 11
GML Geographic Markup Language 10

1. Introduction

1.1. description of the organisation and the department

RWS is an executive organisation of the Ministerie van Infrastructuur en Waterstaat (IenW). RWS maintains and develops the national road network, waterways and waters. The organisation was founded in 1978 and has approximately 8700 people. The dotted red line in figure 1.1 traverses down to the department 'Team toetsing', which is the department where the internship takes place. The 'topo toetsers' are specifically responsible for testing the DTB and BGT geodata.

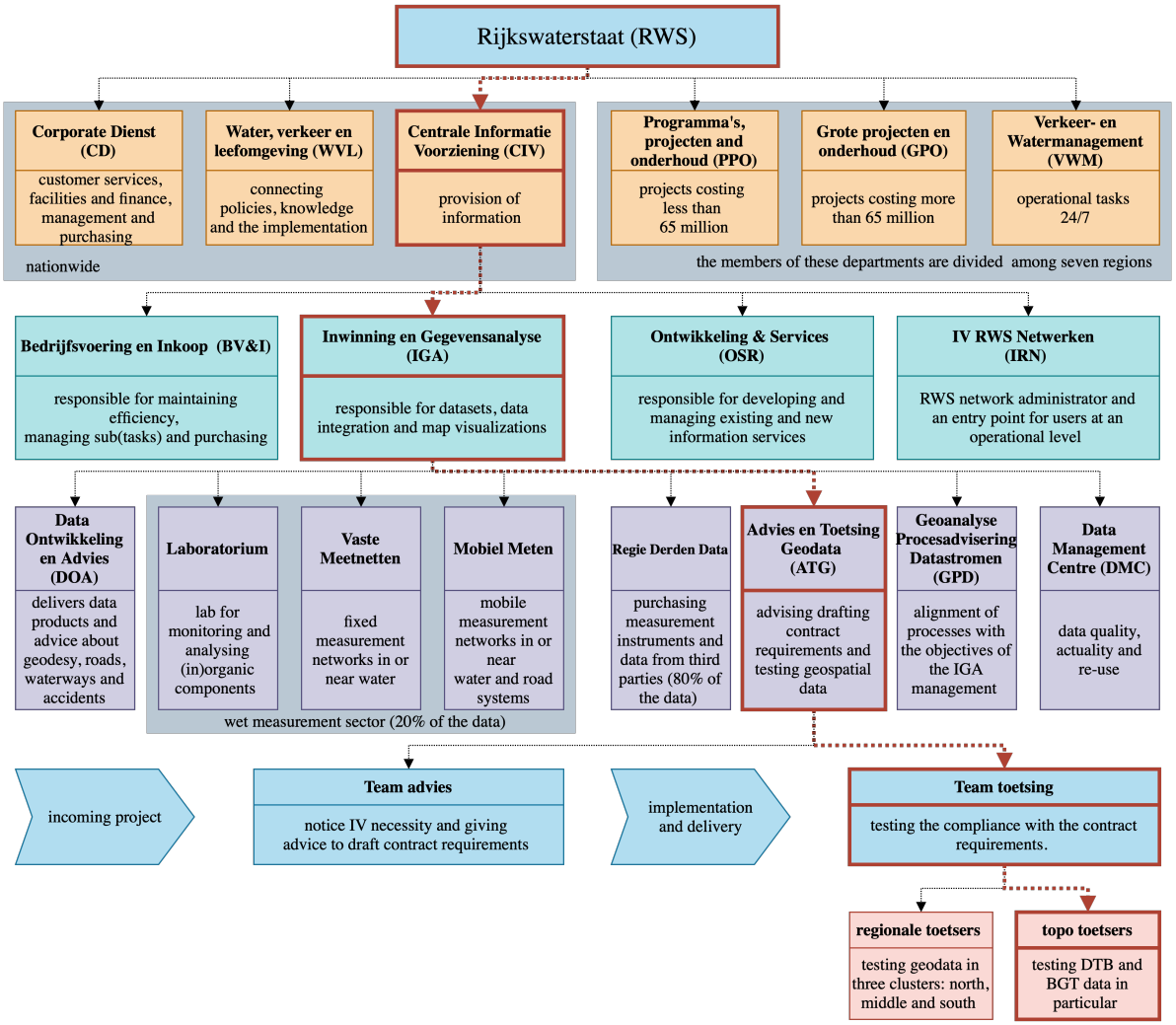


Figure 1.1.: organisation chart

1.2. background

RWS has three obligations on behalf of IenW. RWS' first obligation is to update and deliver BGT data within the RWS data owner area to the Landelijke voorziening (LV)-BGT. Secondly RWS has the obligation to use BGT data within the RWS data owner area. Lastly RWS has the obligation to report errors outside the RWS data owner area through a Mutation Reporting System (MMS) notification via `http://www.verbeterdekaart.nl` [RWS, 2020a].

The BGT locates all physical objects in the Netherlands. It aims to stimulate sharing, to avoid data duplication for instance, and re-using the data. The BGT data is maintained by different data owners [RWS, 2020a]. RWS is the data owner for all national roads and national waterways. The data that is used for the BGT in the RWS data owner area is mainly based on the internal DTB database of RWS. The first delivery of the BGT data inside the RWS data owner area in 2017 is based on the DTB data. Usually the DTB database is also used to process incoming MMS notifications. An incoming MMS notifications indicates that the BGT data is outdated. The process of updating the BGT data could be seen in appendix A.1. The DTB locates objects on and near national roads and national waterways. The data is 2.5 dimensions (includes z-values) and contains points, lines and polygons [RWS, 2020b].

These databases use different data models. A data model describes in which way the objects have to be inserted to ensure consistency and to use the data inter-operable. A data model is a conceptual model that orders data elements and relates them to each other in a standardized way. The BGT uses the data model Informatiemodel geografie (IMGEO). IMGEO is based on the BGT standards and contains required and optional objects. The required objects must satisfy the required quality standards described in figure A.2 [Wouters and Engelsma, 2019]. The optional objects are used to enable data owners to link their own data management system to IMGEO by adding more detailed data. Additionally, these objects have to satisfy certain quality standards [van Rossem, 2012]. The DTB uses another data model, named the DTB-data model, which is applicable to DTB-nat and DTB-droog. DTB-nat contains rivers, shores, coasts and harbors, DTB-droog contains roads and channels [RWS, 2020b]. Both data models are made up of several layers to avoid overlapping polygons in the same layer. On the other hand, ISO- polygons (isovlakken) are allowed to overlap other polygons in the same layer [RWS, 2020b].

Data mapping is the process of creating data element mappings between two distinct data models. The DTB-data model and IMGEO data model are not mapped two-sided. There is a one-sided mapping from the DTB-data model to the IMGEO data model available, but not vice versa. Besides the element mapping, there are also other differences that influence the mapping:

1. Z-value differences: The DTB always contains z-values, while the BGT does not.
2. Layer differences: The DTB did not contain negative layers in the past, this mean that layer one could either be above or underneath the ground level. For example, an underground tunnel would become layer one. On the contrary, the BGT does contain negative layers. The provided solution by RWS is to add a fourth and fifth layer to the DTB in the future. These layers will serve as the negative layers.
3. Quality differences: The accuracy between points of the DTB is higher than the accuracy between points of the BGT

1.3. description of the project

In the first delivery of BGT data inside the RWS data owner area not all attributes could be mapped one-to-one. Therefore, many polygons contained the value 'transitie'. This value had been modified since last year. Although the DTB has still been updated, the BGT has not been updated during that time. A reason for this is that contractors only provided new DTB data and no BGT data. In the future contractors will provide both DTB data and BGT data.

The introduction of the BGT resulted in two related geospatial databases that were not synchronized. It could now be detected whether the data is up-to-date, this is done from the interpretation of aerial

images and based on the dates attached to the objects. Interpreting aerial images automatically would require more advanced techniques such as machine learning. It will therefore be investigated if dates can be used to synchronize the DTB and BGT.

The research will be based on the existing data mapping, this is because two-sided mapping would require more time and it is unsure whether it would be possible at all. The main focus will therefore not be on producing or editing the data mapping, but on the general method and how the individual parts influence the general method. The general decision method aims to decide which database is more up-to-date. This also locates where it is most up-to-date and in this way it could contribute to the synchronization of two geospatial databases. The scope of this research is limited to polygons, because of the overlapping area between the BGT and DTB that could be used. The scope is also limited to the ground layer to avoid overlapping polygons in one of the databases.

The main research question is therefore: *Could dates be used as indicator for the synchronization of two related geospatial databases containing polygons?*

Data mapping must be done in order to relate the data models to each other.

- How is the data mapped?
- How does the data mapping influence the synchronization?

Besides data mapping, the objects in the datasets must be referenced to each other.

- How are the objects of the two related geospatial databases referenced to each other?
- Does the way in which the objects are referenced to each other influence the synchronization?

2. Method

The method is based on three different parts. The first part is based on the data mapping, which influences the conversion from the DTB to the BGT, and the attribute comparison between the databases. The second part is the way in which the objects are referenced to each other. The last part is the date comparison method to decide at which location which database is more up-to-date. Every box in figure 2.1 is explained in more detail in the following sections. The idea of using databases and converting the data to the IMGEO standard with critical attributes is based on a document from RWS [Wouters and Engelsma, 2019]

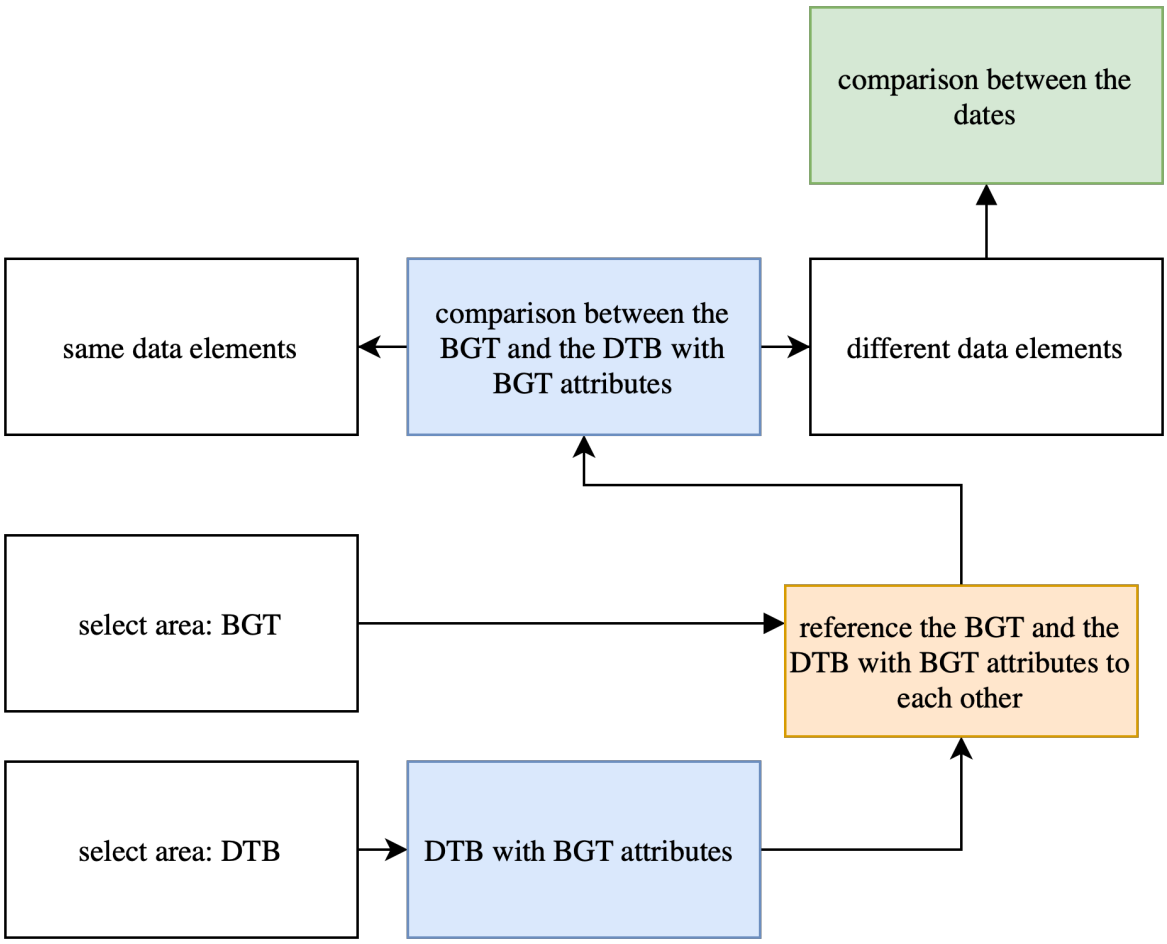
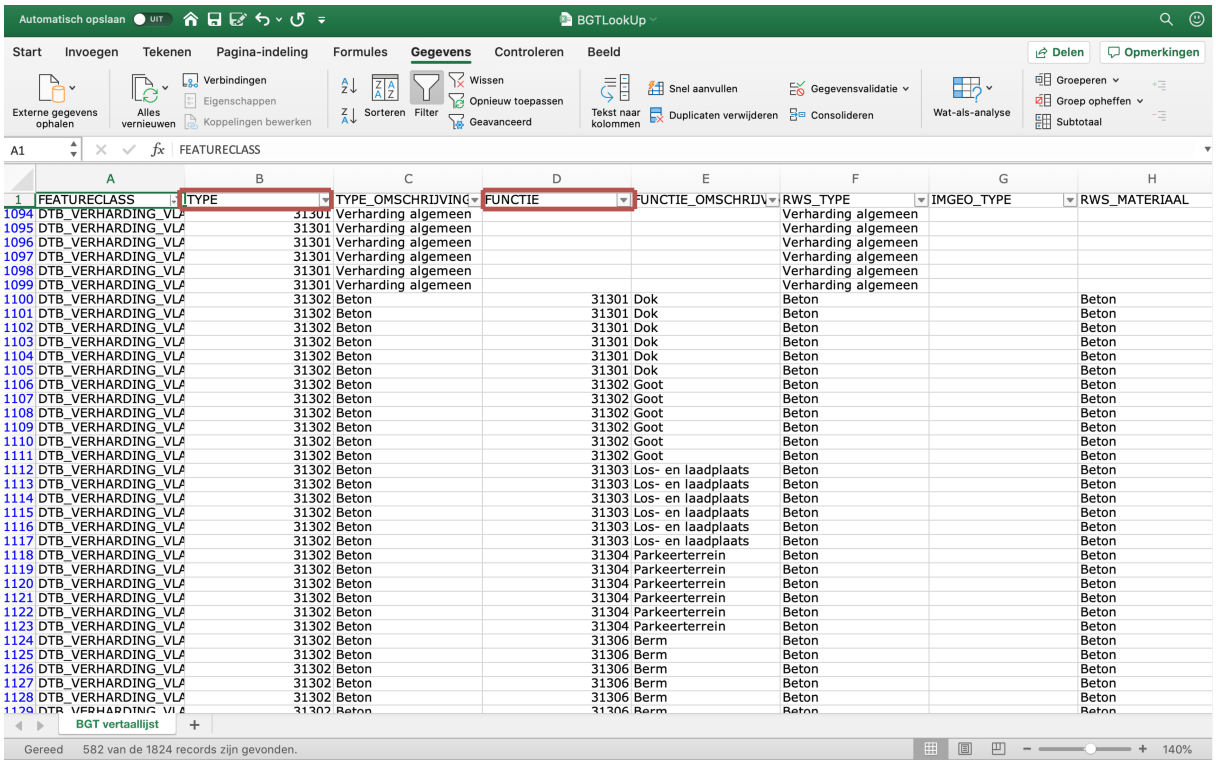


Figure 2.1.: the general method

2.1. select area: BGT and converted DTB

At first, the BGT and DTB objects are stored in two relational databases. In this way, attributes can be added to the objects and the required object can be selected.



	A	B	C	D	E	F	G	H
1	FEATURECLASS	TYPE	TYPE_OMSCHRIJVING	FUNCTIE	FUNCTIE_OMSCHRIJV	RWS_TYPE	IMGEO_TYPE	RWS_MATERIAAL
1094	DTB_VERHARDING_VLA	31301	Verharding algemeen			Verharding algemeen		
1095	DTB_VERHARDING_VLA	31301	Verharding algemeen			Verharding algemeen		
1096	DTB_VERHARDING_VLA	31301	Verharding algemeen			Verharding algemeen		
1097	DTB_VERHARDING_VLA	31301	Verharding algemeen			Verharding algemeen		
1098	DTB_VERHARDING_VLA	31301	Verharding algemeen			Verharding algemeen		
1099	DTB_VERHARDING_VLA	31301	Verharding algemeen			Verharding algemeen		
1100	DTB_VERHARDING_VLA	31302	Beton	31301	Dok	Beton		Beton
1101	DTB_VERHARDING_VLA	31302	Beton	31301	Dok	Beton		Beton
1102	DTB_VERHARDING_VLA	31302	Beton	31301	Dok	Beton		Beton
1103	DTB_VERHARDING_VLA	31302	Beton	31301	Dok	Beton		Beton
1104	DTB_VERHARDING_VLA	31302	Beton	31301	Dok	Beton		Beton
1105	DTB_VERHARDING_VLA	31302	Beton	31301	Dok	Beton		Beton
1106	DTB_VERHARDING_VLA	31302	Beton	31302	Goot	Beton		Beton
1107	DTB_VERHARDING_VLA	31302	Beton	31302	Goot	Beton		Beton
1108	DTB_VERHARDING_VLA	31302	Beton	31302	Goot	Beton		Beton
1109	DTB_VERHARDING_VLA	31302	Beton	31302	Goot	Beton		Beton
1110	DTB_VERHARDING_VLA	31302	Beton	31302	Goot	Beton		Beton
1111	DTB_VERHARDING_VLA	31302	Beton	31302	Goot	Beton		Beton
1112	DTB_VERHARDING_VLA	31302	Beton	31303	Los- en laadplaats	Beton		Beton
1113	DTB_VERHARDING_VLA	31302	Beton	31303	Los- en laadplaats	Beton		Beton
1114	DTB_VERHARDING_VLA	31302	Beton	31303	Los- en laadplaats	Beton		Beton
1115	DTB_VERHARDING_VLA	31302	Beton	31303	Los- en laadplaats	Beton		Beton
1116	DTB_VERHARDING_VLA	31302	Beton	31303	Los- en laadplaats	Beton		Beton
1117	DTB_VERHARDING_VLA	31302	Beton	31303	Los- en laadplaats	Beton		Beton
1118	DTB_VERHARDING_VLA	31302	Beton	31304	Parkeerterrein	Beton		Beton
1119	DTB_VERHARDING_VLA	31302	Beton	31304	Parkeerterrein	Beton		Beton
1120	DTB_VERHARDING_VLA	31302	Beton	31304	Parkeerterrein	Beton		Beton
1121	DTB_VERHARDING_VLA	31302	Beton	31304	Parkeerterrein	Beton		Beton
1122	DTB_VERHARDING_VLA	31302	Beton	31304	Parkeerterrein	Beton		Beton
1123	DTB_VERHARDING_VLA	31302	Beton	31304	Parkeerterrein	Beton		Beton
1124	DTB_VERHARDING_VLA	31302	Beton	31306	Berm	Beton		Beton
1125	DTB_VERHARDING_VLA	31302	Beton	31306	Berm	Beton		Beton
1126	DTB_VERHARDING_VLA	31302	Beton	31306	Berm	Beton		Beton
1127	DTB_VERHARDING_VLA	31302	Beton	31306	Berm	Beton		Beton
1128	DTB_VERHARDING_VLA	31302	Beton	31306	Berm	Beton		Beton

Figure 2.2.: Excel file, LookUpTable, with the attributes: 'type' and 'function' and without the attribute: 'niveau'

The data models are mapped to each other. Data mapping is the creation of data element mappings between the two distinct data models: the IMGEO-data model and the DTB-data model. The mapping makes it possible to compare the objects and to integrate objects of different data models. The DTB are mapped to the BGT, because there is only a one-sided mapping available from the DTB to the BGT. This one-sided mapping is an excel file, the LookUpTable, as could be seen in figure 2.2. A unique row in the excel file is selected to do the data mapping. The row contains DTB attributes, which is used to select the row, and the corresponding BGT attributes, which is added to the DTB object in the database. The DTB attributes, that are used to select a unique row, are:

- type
- function
- niveau

The corresponding BGT attributes, that are added to the object in the database, are:

- `GEOBGT_FCL`
- BGTPLUSTYPE
- FYSIEKVOORKOMEN
- `FUNCTIE_NEW`
- SUBTYPE

After the BGT attributes are added to the DTB object, both the converted DTB objects and the BGT objects adhering to the scope of the research will be exported from the database. The scope of the research requires the following:

- The object must have a geometry type of type Polygon, MultiPolygon or CurvePolygon.
- The object must be located on the ground level. In other words, the object must have niveau zero.

- The object must intersect the area of interest.

Figure 2.3.: overlapping DTB and BGT geometries

The option to select only the objects in the RWS data owner area were considered as well. It was not possible to detect the geometries, where there was no DTB data in the RWS data owner area, as could be seen in figure 2.3 (red part). Consequently, this option were not used.

2.2. referencing the BGT and the converted DTB

The BGT and DTB exists of multiple individual layers. Every layer contains its own type of objects such as roads or waters. The individual thematic BGT and DTB layers are merged to begin with. This simplifies the next operations.

There were two methods considered to reference the converted DTB and the BGT to each other.

In general, it will be most efficient to join objects. Objects can be joined using unique Identifier (ID)s. This could be done using the primary key, in this case`dtb_id` or `bgt_id`, which can be added as a foreign key to the corresponding object.

Corresponding to this, the first method figures out which objects belong together. The difficulty were however to determine which objects had to belong together. This were determined based on the overlapping area. If the overlapping area were higher than 60 percent according to the DTB or BGT area, the objects had to belong together. In theory, this means that one DTB object could reference to multiple BGT objects and vice versa. This could only be the case when the layer contains overlapping polygons. Although the references between the objects are preferred, objects, that do not correspond to another object for more than 60 percent, could not be referenced. As a consequence, no attribute comparison could be made, because these objects would become blue or green area as could be seen in figure 2.3. On top

of that, an overlap of 60 percent does not guarantee that the corresponding object is found. Lastly, the interpretation of many-to-many references can be difficult. Especially, when the references are uncertain. Ideally, the objects would be joined on their IDs. However, it requires some certainty about which objects belong together and which objects are new. Therefore, it were decided not to use this method.

Therefore, the other option, a union, were considered as well. A union is used to compare the attributes per geometry as could be seen in figure 2.4. New geometries are created by a union. The method has two advantages: the corresponding object does not have to be known and the method is relatively fast.

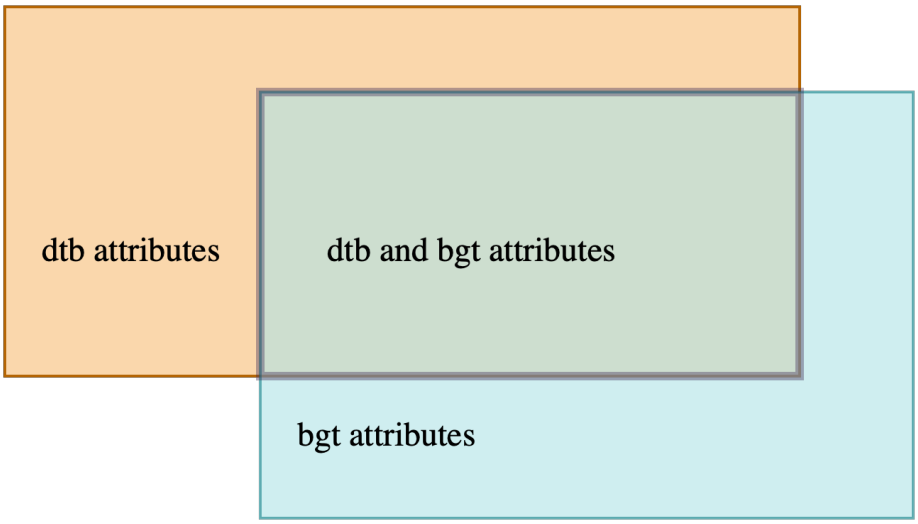


Figure 2.4.: union DTB and BGT

2.3. comparison between the BGT and the converted DTB

The attributes could only be compared at the yellow area, where there is BGT and DTB data, as could be seen in figure 2.3. The following BGT attributes were used for the comparison:

- imgeotype
- fysiekvoorkomen
- bgtplustype
- functie

The DTB must always contain BGT data, where RWS is the data owner. With this in mind, the red area in figure 2.3 indicates that the DTB or BGT is outdated. However, the attributes and dates can not be compared, because there is no DTB data present to compare it with.

2.3.1. comparison between the dates

Next, at those locations, where the attributes differ, the dates are compared. The life cycle of the dates can be seen in figure 2.5. This life cycle is based on the following knowledge obtained from RWS. The DTB mutation date will be updated for small mutations. However, the DTB collection date will be updated for large mutations, because a new object will be created. Because of this, it were assumed that the following dates could be compared to each other:

- LV publication date or the beginning of the BGT formal registration date = DTB mutation date

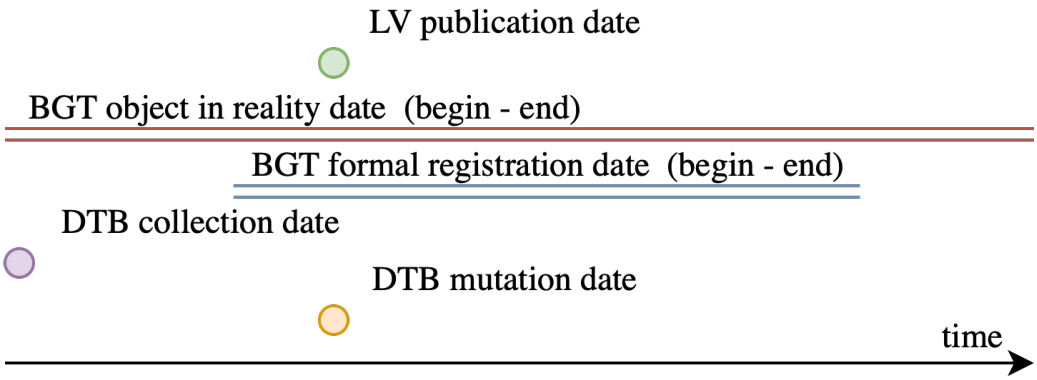


Figure 2.5.: lifecycle of an object

- beginning of the BGT object in reality date = DTB collection date

Because these dates could differ during a certain time, a threshold of 180 days were implemented. This threshold were used, because objects such as buildings and roads have to be up-to-date within 6 months. According to the related dates and the knowledge obtained from RWS, two deviations have been considered:

1. Is it the another object?
2. Is it the another mutation?

These options were implemented in the method to compare BGT and DTB dates as could be seen in figure 2.6. If is it not the same object or mutation despite the threshold of 180 days, it is investigated which object is more up-to-date.

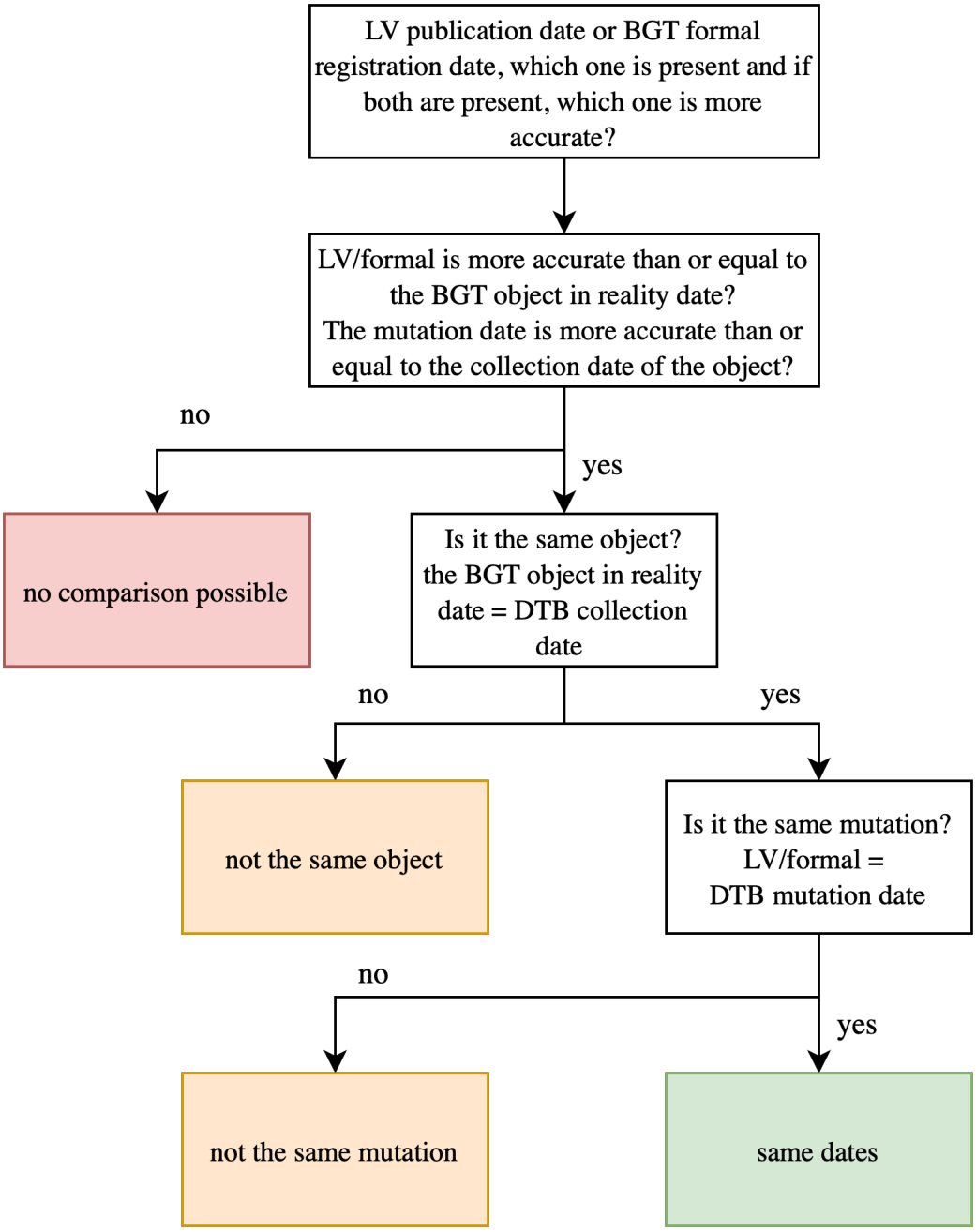


Figure 2.6.: method to compare BGT and DTB dates

3. Data and software

The url of the GitHub repository is `https://github.com/kjstaring/datesynchronization`

3.1. data

layer	[van Rossem, 2012]
bgt_bak	points
bgt_begroeidterreindeel	polygons
bgt_bord	points
bgt_buurt	registration area
bgt_functioneelgebied	
bgt_gebouwinstallatie	polygons
bgt_installatie	points
bgt_kast	points
bgt_kunstwerkdeel	polygons, multipolygons, lines
bgt_mast	points
bgt_onbegroeidterreindeel	polygons
bgt_ondersteunendwaterdeel	polygons
bgt_ondersteunendwegdeel	polygons
bgt_ongeclassificeerdobject	polygons
bgt_openbareruimte	registration area
bgt_openbareruimtelabel	registration area
bgt_overbruggingsdeel	polygons
bgt_overigbouwwerk	polygons, multipolygons
bgt_overigescheiding	lines, polygons
bgt_paal	points
bgt_pand	polygons, multipolygons
bgt_put	points
bgt_scheiding	lines, polygons
bgt_sensor	points
bgt_spoor	lines
bgt_straatmeubilair	points
bgt_tunneldeel	polygons
bgt_vegetatieobject	points, lines or polygons
bgt_waterdeel	polygons
bgt_waterinrichtingselement	points, lines
bgt_wegdeel	polygons
bgt_weginrichtingselement	points, lines or polygons
bgt_wijk	registration area

Table 3.1.: The purple BGT layers are used

The data folder in the github repository contains BGT test data and the exel file, which is the LookUpTable provided by RWS. This data has been downloaded from Publieke Dienstverlening Op de Kaart (PDOK) in Geographic Markup Language (GML) format. There are three test datasets:

name	location	size
BGT	A27	9.7 MB
BGT2	Zeeland	24.3 MB
BGT3	Rotteram	178 MB

Table 3.2.: BGTtest datasets

The DTB data, which is not in the data folder, has been provided by RWS as an ESRI FileGDP. The DTB has 16 different layers of which only 6 contribute to the ground level area according to the RWS data owner area. These layers are purple in figure 3.3. The `overige_vlakken` must only be included in the BGT if it is a flood defence system named 'kering' in dutch.

layer	usage	description
`dtb_bebouwing_vlakken`	converted	buildings
`dtb_bekleding_vlakken`	converted	constructed top layer not used by traffic
`dtb_grond_vlakken`	converted	types of landuse
`dtb_installatie_vlakken`	converted	installations such as a mills or walls
`dtb_kunstwerk_vlakken`	not available on the ground level	
`dtb_markering_vlakken`	overlapping polygons	
`dtb_overige_vlakken`	overlapping polygons	
`dtb_terrein_vlakken`	not available in area as polygons	
`dtb_verharding_vlakken`	converted	surfacing
`dtb_water_vlakken`	converted	water
`dtb_weg_vlakken`	overlapping polygons	

Table 3.3.: the purple DTB layers are used

The DTB data could be mapped to the BGT with the excel file. The excel fileis a one-sided mapping from the DTB to the BGT provided by RWS. I have changed 'oever' and 'slootkant' in the excel file.

3.2. software

ArcGIS is a commercial Geographic Information System (GIS). In ArcGIS, ESRI FileGDP are used when the attribute table of an individual shapefile exceeds the data size of 2GB. Since ArcGIS is commercial, the open source competitor QGIS is used. QGIS has standard read capabilities for ESRI FileGDPs, but no standard write capabilities. Therefore, additional software has to be installed, which is Osgeo4mac for Mac OSX or Osgeo4W for Windows. This software is needed to handle ESRI FileGDPs in PostgreSQL as well. PostgreSQL is an open-source relational database. Besides PostgreSQL and QGIS, the programming language Python is used to access PostgreSQL.

Installations used for the insertion of an ESRI FileGDP in PostgreSQL:

- brew tap osgeo/osgeo4mac
- brew install osgeo-postgresql
- brew install osgeo-gdal

- brew install osgeo-gdal-filegdb
- brew install osgeo-postgis

Installed and used python modules:

- pip3 install psycopg2
- pip3 install pandas
- pip3 install xlrd

4. Implementation

4.1. select area: BGT and converted DTB

4.1.1. DTB

```
#!/bin/bash
cd /Users/karinstaring/Documents/rijkswaterstaat/implementation/scripts/mapping
# create databatase with postgis and sfcgal extension in Python
python3 ./create_DTBdatabase.py
# import PIVRI.gdp in PostgreSQL
ogr2ogr -progress -overwrite -f "PostgreSQL" PG:"host=localhost user=postgres dbname=dtbdatabase password=1234"
# data mapping, output: DTB with BGT attributes = converted DTB
python3 ./mapping_DTBdatabase.py
# delete and rename attributes
python3 ./change_DTBattributes.py
cd /Users/karinstaring/Documents/rijkswaterstaat/implementation/shapefiles/dtb_shapesfiles
# iterature over DTB tables and export selected objects
for VARIABLE in dtb_bebouwing_vlakken dtb_bekleding_vlakken dtb_grond_vlakken dtb_installatie_vlakken dtb_overig
do
    ogr2ogr -f "ESRI Shapefile" /Users/karinstaring/Documents/rijkswaterstaat/implementation/shapefiles/dtb_shap
done
```

Figure 4.1.: create and export the converted DTB. The ogr2ogr lines are longer than the lines in the figure.

At first, the DTB objects are mapped to BGT objects. As could be seen in figure 4.1, Python is used to create a database in PostgreSQL with the extensions postgis and `postgis_sfcgal` for the DTB. The Python script explains how databases are created with extensions is added in appendix A.3. After the creation of the database, the bash file is used to execute a command that imports the DTB ESRI FileGDP, PIVRI.gdp, in PostgreSQL as could be seen in figure 4.1.

```
#1. DTB mapped to BGT
conn = psycopg2.connect("host=localhost dbname= dtbdatabase user=postgres password=1234")
cur = conn.cursor()

layers = ['dtb_bebouwing_vlakken', 'dtb_bekleding_vlakken',
'dtb_grond_vlakken', 'dtb_installatie_vlakken', 'dtb_overige_vlakken',
'dtb_verharding_vlakken', 'dtb_water_vlakken']

#2. Iterate over all layers
for layer_name in layers:
    print(layer_name)
    try:
        mapping_dtb(layer_name)
    except:
        print('layer not available in DTB: ' + str(layer_name))
        pass

print('finished mapping')
conn.close()
```

Figure 4.2.: the function `mapping_dtb` is called for every layer

Subsequently, Python is used to make a database connection with the module psycopg2 and the DTB objects are mapped to BGT objects. The data mapping is done for every layer of table 3.3 including `dtb_overige_vlakken` as could be seen in figure 4.2. It could be that `dtb_terrein_vlakken` has to be included as well, but this has not been done. The `data_mapping` function could be seen in appendix A.4. A unique row in the excel file is selected. The row contains DTB attributes, which are used to select the row, and the corresponding BGT attributes, which are added to the DTB object.

Next, Python is also used to delete and rename attributes for every DTB layer. Although this seems confusing at first, it became more clear which attributes belong to the DTB object after the union. The Python script, which is used to delete and rename DTB attributes, could be seen in appendix A.5. Additionally, the renamed DTB attributes could be seen in table 4.1.

old name DTB attribute	added BGT attribute	new name attribute
dtb_id		dtb_id
niveau		dtb_niveau
datum_mutatie		dtb_datum1
datum_inwinning		dtb_datum2
type		dtb_type
functie		dtb_oldf
bronhouders		dtb_bronh
shape		dtb_geom
	geobgt_fcl	dtb_geobgt
	bgtplustyp	dtb_plus
	fysiekvoor	dtb_fys
	functie_new	dtb_func
	subtype	dtb_sub

Table 4.1.: The renamed DTB attributes

Lastly, the bash file iterates over every table in the DTB database. Only the objects, that have the following characteristics, are exported as ESRI shapefiles:

- located on the ground level, where ('dtb_niveau' = 0
- intersect the area of interest. The area of interest is conducted using a LineString with reference system 28992.
- the geometry type is Polygon, MultiPolygon or CurvePolygon.

4.1.2. BGT

Firstly, a Python script is used to create a BGT database in PostgreSQL with the extensions postgis and `postgis_sfcgal` in approximately the same way as appendix A.3. After the creation of the BGT database, the bash file is used to execute a command that imports the BGT shapefiles in PostgreSQL as could be seen in figure 4.3.

Next, a Python script is used to delete and rename attributes for every BGT layer in table 3.1. This has been done in approximately the same way as appendix A.5. Only, the attributes, that are renamed and deleted, differ. The renamed BGT attributes could be seen in table 4.2.

Lastly, the bash file iterates over every table in the BGT database. Only the objects, that have the following characteristics, are exported as ESRI shapefiles:

- located on the ground level, where ('dtb_niveau' = 0
- intersect the area of interest. The area of interest is conducted using a LineString with reference system 28992.

```
cd /Users/karinstaring/Documents/rijkswaterstaat/implementation/scripts/mapping
# create databatase with postgis and sfcgal extension in Python
python3 ./create_BGTdatabase.py
# import BGT gml files from folder (here: BGT) in PostgreSQL
cd /Users/karinstaring/Documents/rijkswaterstaat/implementation/data/BGT
for f in *.gml;
do
    base=${f%.gml}
    ogr2ogr -progress -overwrite -f "PostgreSQL" PG:"host=localhost user=postgres
done
# delete and rename attributes
cd /Users/karinstaring/Documents/rijkswaterstaat/implementation/scripts/mapping
layerlist=`python3 ./change_BGTattributes.py`
# iterature over BGT tables and export selected objects
cd /Users/karinstaring/Documents/rijkswaterstaat/implementation/shapefiles/bgt_sh
for VARIABLE in bgt_begroeidterreindeel bgt_kunstwerkdeel bgt_onbegroeidterreinde
do
    ogr2ogr -f "ESRI Shapefile" /Users/karinstaring/Documents/rijkswaterstaat/imp
done
```

Figure 4.3.: create and export the BGT. The ogr2ogr lines are longer than the lines in the figure.

old name BGT attribute	new name attribute
gml_id	bgt_id
relatievehoogteligging	bgt_niveau
creationdate	bgt_datum1
lv_publicatiedatum	bgt_datum2
tijdstipregistratie	bgt_datum3
lokaalid	bgt_lokaal
terminationdate	bgt_datum4
eindregistratie	bgt_datum5
bronhouder	bgt_bronh
wkb_geometry	bgt_geom
bgt_type	bgt_plus
class	bgt_fys1
bgt_fysiekvoorkomen	bgt_fys2
surfacematerial	bgt_fys3
function	bgt_func

Table 4.2.: The renamed BGT attributes

- the geometry type is Polygon, MultiPolygon or CurvePolygon.

4.2. referencing the BGT and the converted DTB

Two methods have been implemented: the union and the method based on the overlapping area. The individual thematic BGT layers and individual thematic DTB layers have to be merged in QGIS to perform a union and to insert all objects in one table.

Firstly, the ESRI shapefiles have been processed in QGIS, because this part has not been automated yet. The data management tool 'merge vector layers' in QGIS has been used to merge layers. Although the layers did not contain objects with points or LineStrings, layers, that were LineString-based, had to

be removed. The geoprocessing tool 'union' were used to create the union with the merged BGT and merged DTB layer as input. After the union were performed, it were important to check if the attribute 'layer' belonged to the BGT and not to the DTB. Otherwise, the comparison would have used the attribute incorrectly later.

```
#!/bin/bash
cd /Users/karinstaring/Documents/rijkswaterstaat/implementation/scripts/comparison
python3 ./create_MERGEDdatabase.py

ogr2ogr -f "PostgreSQL" PG:"host=localhost user=postgres dbname=mergeddatabase password=1234" -nlt PROMOTE_TO_MULTI
ogr2ogr -f "PostgreSQL" PG:"host=localhost user=postgres dbname=mergeddatabase password=1234" -nlt PROMOTE_TO_MULTI
ogr2ogr -f "PostgreSQL" PG:"host=localhost user=postgres dbname=mergeddatabase password=1234" -nlt PROMOTE_TO_MULTI
```

Figure 4.4.: create database and import merged DTB, merged BGT and the union. The ogr2ogr lines are longer than the lines in the figure.

Python is used to create a merged database in PostgreSQL with the extensions postgis and `postgis_sfcgal`. Three shapefiles are imported in the database as could be seen in figure 4.4, namely the merged BGT layer, the merged DTB layer and the union layer, which is created by the union.

The union creates new geometries, which contain the overlapping attributes from the BGT and the DTB. However, it were more complicated for the method based on the overlapping area, because the corresponding object had to be found as could be seen in figure 4.5. Besides this function, the opposite function had to be called `use_dtb_geometries`. These functions could be seen in the github repository, but they will not be further explained. The reason for this is that the corresponding object could not be found for many objects and therefore no comparison between the objects is possible.

```
def use_bgt_geometries():
    bgt_vlakken = """select st_astext(wkb_geometry), bgt_
    cur.execute(bgt_vlakken)
    bgt = cur.fetchall()

    add_compared = """ALTER TABLE public.bgt_merged ADD C
    cur.execute(add_compared)

    add_compared = """ALTER TABLE public.bgt_merged ADD COLUMN IF NOT EXISTS dtb_datum1 date"""
    cur.execute(add_compared)

    add_compared = """ALTER TABLE public.bgt_merged ADD COLUMN IF NOT EXISTS dtb_datum2 date"""
    cur.execute(add_compared)

    k_bgt = 0
    bgt_count = len(bgt)

    for obj_bgt in bgt:

        print("{:.2f}".format((k_bgt / bgt_count) * 100), end='\r')
        k_bgt = k_bgt + 1

        overlap = """select dtb_merged.ogc_fid, (st_area(st_intersection(st_setsrid('{}'::geometry, 900914), dtb_merged.wkb_geometry))
        /st_area(st_setsrid('{}'::geometry, 900914))), dtb_merged.dtb_geobgt from public.dtb_merged
        where (st_area(st_intersection(st_setsrid('{}'::geometry, 900914), dtb_merged.wkb_geometry))/st_area(st_setsrid('{}'::geometry, 900914))) > 0.6""".format(obj_bgt[0],obj_bgt[0], obj_
        cur.execute(overlap)
        overlappend_vlak = cur.fetchall()

        number_vlakken = len(overlappend_vlak)
        if number_vlakken > 0:
            percentage = 0
            index = 0
            if number_vlakken > 1:
                for i in range(number_vlakken):
                    print(overlappend_vlak[i])
                    if overlappend_vlak[i][1] > percentage:
                        percentage = overlappend_vlak[i][1]
                        index = i
            object_dtb = """select dtb_id, dtb_geobgt, dtb_fys, dtb_plus, dtb_func, dtb_datum1, dtb_datum2 from public.dtb_merged where ogc_fid = '{}'""".format(overlappend_vlak[index][0])
```

select the geometries of all BGT objects in the table: bgt_merged

plus, bgt_func, ogc_fid from public.bgt_merged"""

iterate over the BGT objects

select the DTB objects where the geometry overlaps the BGT geometry for more than 60 percent

If more than one DTB object is selected, which is strange, the object with the highest overlap will be used as corresponding object.

Figure 4.5.: function called `use_bgt_geometries`

4.3. comparison between the BGT and the converted DTB

The function in appendix A.6 iterates over all objects, which were created by the union. The function `compare_attributes` is called for every object. This function returns a category value, that is assigned to the object' column category as could be seen in appendix A.6. The attributes that are compared are shown in table 4.3.

attribuut	merged DTB	merged BGT
imgeotype	dtb_geobgt	layer
fysiekvoorkomen	dtb_fys	bgt_fys1 bgt_fys2 bgt_fys3
bgtplustype	dtb_plus	bgt_plus
functie	dtb_func	bgt_func

Table 4.3.: compared attributes

The function, which is used to compare the attributes, is shown in appendix A.7. In this way, the following values are returned by the function and are assigned to the objects' column category:

1. These values are only based on the unique bgt_id and unique dtb_id:
   - no_bgt_dtb: no BGT and DTB data
   - no_bgt: no BGT data
   - no_dtb: no DTB data
2. This value is also based on the attribute bgt_bronh where bgt_bronh is 'L0002', which is the code for IenW. RWS is an executive organisation of IenW:
   - rws_no_dtb: there is no DTB data in RWS data owner area
3. The attributes are also compared based on the imgeotype, fysiekvoorkomen, bgtplustype and functie as could be seen in table 4.3 and in appendix A.7:
   - dtb_bgt_no_layer: no BGT layer available
   - dtb_bgt_no_geobgt: no DTB layer available, because the object could not be converted to an imgeotype
   - onbekende_waarden: the BGT attribute contains values from table 4.4
   - different: different attributes
   - same: all the attributes are the same

The attributes are also considered to be different if one attribute has a None value, while the other has not.

old name BGT attribute	new name attribute	value
class	bgt_fys1	oever, slootkant
class	bgt_fys1	watervlakte
class	bgt_fys1	landhoofd
class	bgt_fys1	waardeOnbekend
class, bgt_fysiekvoorkomen and surfacematerial	bgt_fys1, bgt_fys2 and bgt_fys3	transitie
bgt_type	bgt_plus	transitie
bgt_type	bgt_plus	niet-bgt
function	bgt_func	ruiterpad
function	bgt_func	voetgangersgebied
function	bgt_func	overweg
function	bgt_func	transitie
function	bgt_func	niet-bgt

Table 4.4.: values not available in the lookup table: onbekende_waarden

4.3.1. comparison between the dates

At the location where the category value is different, `dtb_bgt_no_later` or `dtb_bgt_no_geobgt`, the method to compare BGT and DTB dates from figure 2.6 is used. The method is implemented in Python and could be seen in appendix A.8. The function returns a `synchronization_value`, which is assigned to the object' column sync. The following values are assigned:

- `not_possible`
- `dtb_object_more_accurate`
- `bgt_object_more_accurate`
- `dtb_mutation_more_accurate`
- `bgt_mutation_more_accurate`
- `same_object_same_mutation`

Finally, the bash file is used to export the table as a ESRI shapefile. The added attributes, category and sync, can be visualised in QGIS. Figure 4.6 shows how this can be done. The used styles can be downloaded from the GitHub repository.

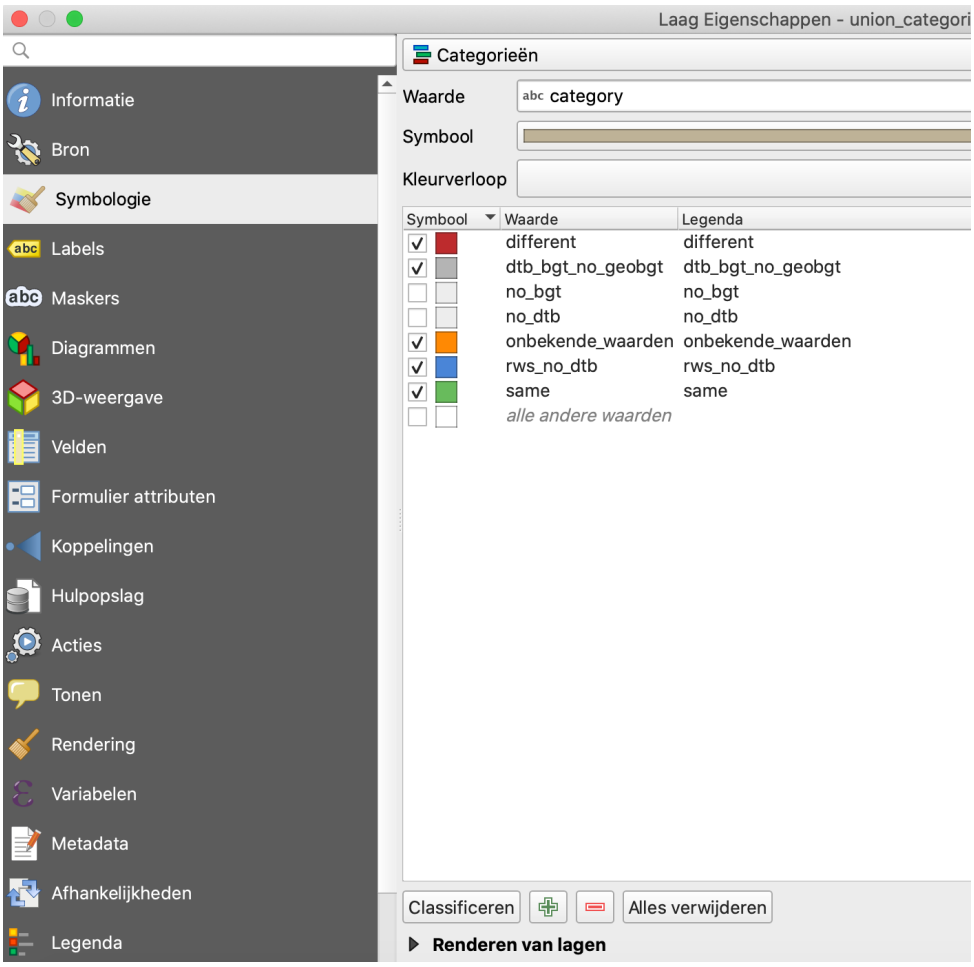


Figure 4.6.: visualisation of the categories in QGIS

5. Results

The method were tested with the BGT test data from table 3.2. There were no real results, because no ground truth were used. Instead this section will be used for some general remarks.

5.1. incorrect DTB data mapping

As could be seen in figure 5.2 and figure 5.3, almost all cases contain the category values different or `dtb_bgt_no_geobgt`. An exception is figure 5.1, because the DTB values have been modified according to the excel file. Although the `dtb_bgt_no_geobgt` value is still present, it means that the DTB attributes could not be mapped to a row in the excel file as could be seen in table **??**. The excel, LookUpTable, most likely differs from the conversion, that is used to update the BGT. As a consequence, most objects could not be mapped to a row in the excel file or they were mapped to a different row with different attributes. For this reason, they differ from the corresponding BGT object.

dtb_type	dtb_oldf	**reason**
30304	30301	30304 does not exist with 30301
30305	30301	30305 does not exist with 30301
30307	30301	30307 does not exist with 30301
30314	30302	special character in name
31004		31004 does not exist
31307	30301	31307 does not exist with 30301
31314	31313	31314 does not exist with 31313
31315	31316	31315 does not exist with 31316
31319	31313	31319 does not exist with 31313
31330	31313	31330 does not exist with 31313

Table 5.1.: some reasons that objects get the value `dtb_bgt_no_geobgt`

5.2. 180 days threshold seems too low

Most BGT objects seem more up-to-date than the DTB objects as could be seen in figure 5.4 and figure 5.5. This is unlikely, because DTB objects are used to establish the BGT in the RWS data owner area. The threshold of 180 days is probably too low, because some objects such as buildings and roads have to be up-to-date within 6 months, but other objects have to be up-to-date with 18 months for instance. Another reason, which is not investigated, could be that it is caused by the first delivery in 2017. While the DTB objects are older and converted to BGT objects, the BGT objects got a new date and they are therefore more up-to-date.

5.3. multiple overlapping BGT objects at the ground level

Multiple overlapping BGT objects at the ground level are shown in figure 5.6 and 5.7. When turning a category off, another category becomes visible. This means that there are multiple objects at the ground

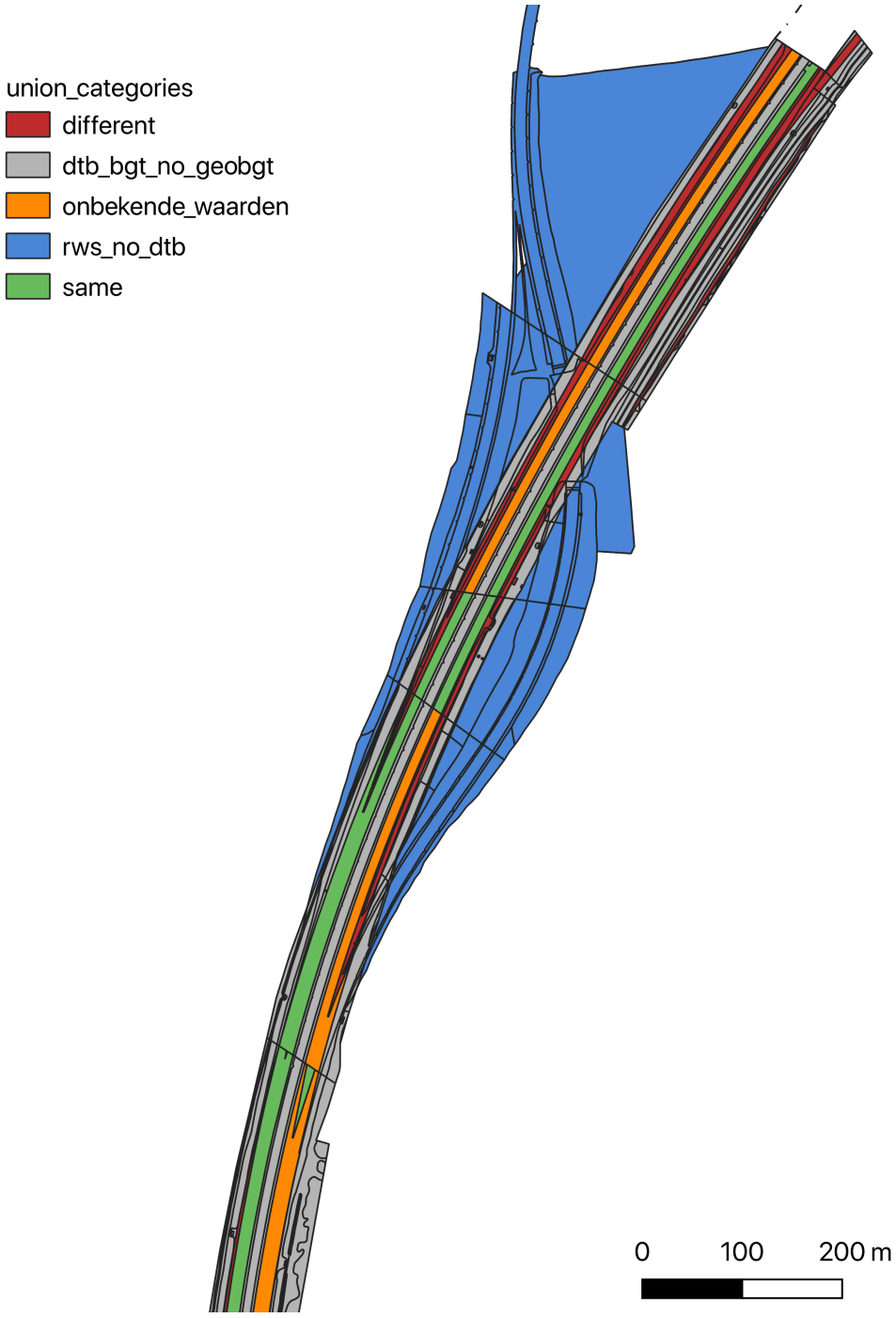


Figure 5.1.: union categories BGT: A27

level compared to each other. The reason for this became visible while implementing the method based on the overlap. In some cases, one DTB object could overlap for more than 60 percent with multiple BGT objects.

Figure 5.2.: union categories BGT2: Zeeland

Figure 5.3.: union categories BGT3: Rotterdam

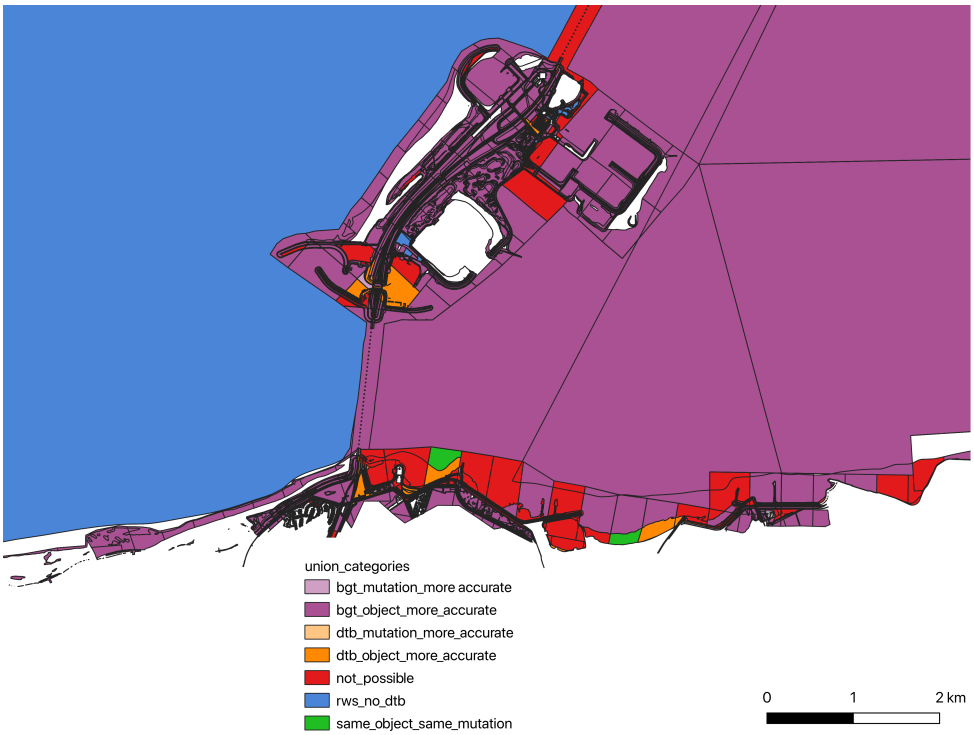


Figure 5.4.: union sync BGT2: Zeeland

Figure 5.5.: union sync BGT3: Rotterdam

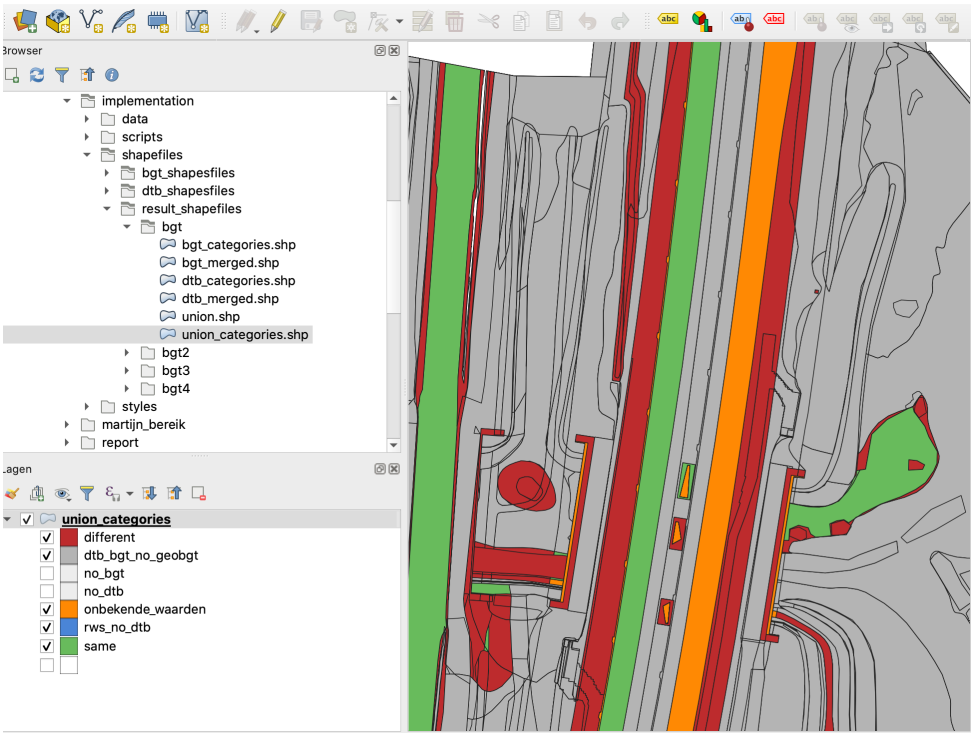


Figure 5.6.: screenshot QGIS with same on BGT: A27

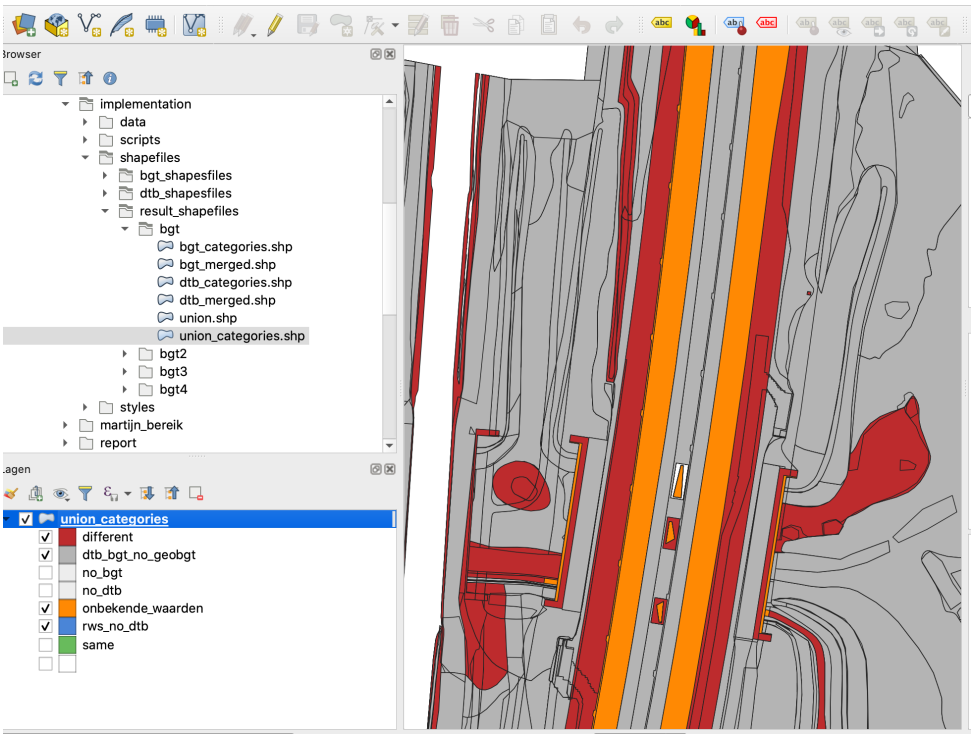


Figure 5.7.: screenshot QGIS with same off BGT: A27

6. Conclusion

RWS has to maintain two related geospatial databases both with their own data model. The first delivery of BGT data inside the RWS data owner area in 2017 has been based on the DTB. As a result of this, the two related geospatial databases became asynchronous which indicates that the updates are not done in both databases.

Because both related geospatial databases receive updates and are large databases, one could argue that it is impossible to maintain these geospatial databases manually. Automatically synchronizing these databases in an optimal way would require a two-sided data mapping. There is however, only a one-sided mapping available from the DTB to the BGT, which will be used for this research.

The accuracy of the objects in the databases could be used to automatically synchronize the databases. The accuracy could be detected from interpreting aerial images or from the dates attached to the object. Because the interpretation of aerial image would require more advanced techniques such as machine learning, we decided to use the dates attached to the object in the database.

The scope of the research is limited in the following ways:

1. The available one-sided data mapping is used.
2. Only polygons is included, lines and points are not.
3. Only the ground layer is included to avoid overlapping polygons in the individual databases.

With this in mind, together with the problem statement, it resulted in the following main research question: **'Could dates be used as indicator for the synchronization of two related geospatial databases containing polygons?'** This research question will be answered using the sub-questions.

In order to answer the sub-questions, a general research method were established. The DTB objects were mapped to the BGT objects in order to make a comparison with the BGT. In order to make a comparison, the converted DTB objects had to be referenced to the corresponding BGT objects and vice versa. After that, their attributes had to be compared to determine which objects are different from each other. If they were different, it has been investigated which object were more up-to-date by comparing the dates. This information could be used for synchronizing the databases.

The implemented conversion and the comparison between the two related geospatial databases could give an answer to the first sub-question **'How is the data mapped?'**. The DTB is mapped to the BGT, because there was only a one-sided mapping available from the DTB to the BGT. Another reason to use this mapping for the comparison were that the BGT in the RWS data owner area is indeed conducted from the DTB. This one-sided mapping is an excel file (name of the file:'LookUpTable'). To do the data mapping, a unique row in the excel file has to be selected. The row contains DTB attributes, which could be used to select the row, and the corresponding BGT attributes, which could be added to the DTB object. In this way the converted DTB with BGT attributes could be compared to the BGT.

Because the comparison could influence the outcome of the method, the second sub-question **'How does the data mapping influence the synchronization?'** has to be answered. This question was answered during the implementation of the method. The data mapping determines if objects are the same and if the dates must be compared. During the implementation, it was discovered that many DTB objects could not be mapped to a row in the excel file, or the objects were mapped to a row/object with different attributes than the attributes of the corresponding BGT object. An explanation for this could be that the excel file is not used directly to update the BGT or/and for the first delivery of the BGT in 2017, but another mapping has been used. The converted DTB objects differ therefore in almost all cases from the BGT objects, which influences the synchronization, because if objects differ, the dates will be compared in the established method.

However, in order to compare the attributes, the objects had to be referenced to each other. How the object are referenced is treated within the third sub-question **'How are the objects of the two related geospatial databases referenced to each other?'**. Two options have been considered, either if the corresponding objects must be referenced to each other based on the overlapping percentage of the objects, or if a union must be used. The main reason for not using the first option is that there were many objects with less than 60 percent overlap, which means that the object could not be referenced to another object, and therefore the attributes of the objects, could not be compared. Another limitation is that the threshold of 60 percent is ambiguously chosen and not based on the accuracy between points for instance. When the accuracy between points would have been incorporated, the accuracy would likely be even higher, resulting therefore in even less objects that could be referenced to each other. Because of these limitations, the second option is considered. The union could be used to compare the attributes for geometries where the two related geospatial databases overlap. The union could be ambiguous as well, because the BGT contains multiple overlapping polygons at the ground level. The reason behind these overlapping polygons is not established in this research. On the other hand, an advantage of the union is that the attributes could be compared for every geometry where the databases overlap, and therefore no corresponding object has to be found. Because of this, the union is used in this research.

To further explain the influence of the union, the fourth sub-question **'Does the way in which the objects are referenced to each other influence the synchronization'** will be answered. The union influences the synchronization in the way that the geometries could not be compared anymore, but it shows for which geometries, that were made by the union, the attributes differ. Because objects are not directly selected with a union, the synchronization itself might be more challenging. It will require further processing. A solution that has not been implemented yet could be to select `dtb_id` and `bgt_id` of the geometry made by the union and to use these IDs to select the original objects in the database. This could be done for those geometries where the DTB or BGT is outdated. In this way the outdated objects could be selected in the database and synchronized afterwards.

Unfortunately, the main research question can not be answered with certainty, because of multiple reasons:

1. The used data mapping to compare the attributes, the excel file, most likely influences the method negatively.
2. Synchronizing the two related geospatial databases requires further processing as stated earlier.
3. The results are not compared with the ground truth. The BGT is normally used as the ground truth.

The used data mapping gives the result that almost all converted DTB objects differ from the BGT objects. Because of this, the dates of almost all objects are compared to each other. It is not investigated if dates could be used as indicator without using data mappings, because correct dates could contain wrong mappings and wrong dates could contain correct mappings. However, if there is no appropriate data mapping, it could be worth investigating. Next, most BGT objects seem more up-to-date than the DTB objects. This is unlikely, because DTB objects are used to establish the BGT in the RWS data owner area. The threshold of 180 days is probably too low, because some objects such as buildings and roads have to be up-to-date within 6 months, but other objects have to be up-to-date with 18 months for instance. We could therefore conclude that dates might be usable in the future, but it could not be conducted from this research.

7. Recommendation

Further research is needed to discover if it is possible to synchronize two related geospatial databases using dates. Recommendations for further research are therefore given in the list below:

- Research related to data mapping
  - A FME script, which contains a conversion from the DTB to the BGT, is used to update the BGT. This FME script might also be used for the first delivery of the BGT in 2017. It has been concluded from this research that it is likely that the FME script used differs from the excel file, the LookUpTable. This could be investigated in further research.
  - Because contractors will provide both DTB and BGT data in the future, the data mapping will be done by every individual contractor. This might result in inconsistent data mappings.
- Research related to object references
  - It could be investigated if the `dtb_terein_vlakken` must be included as well, because this is not done.
  - The used references for both referencing methods were sometimes ambiguous due to the fact that the same DTB object could correspond to multiple BGT objects at the same location. The reason for this is that the BGT contains multiple overlapping polygons at the ground level. This results in multiple references at the same location, which results in different comparisons at the same location. This makes the outcome of the comparison ambiguous. Therefore, the reason behind the overlapping polygons at the ground level in the BGT has to be further explored.
  - Although the union is used instead of the method based on the overlap, the method based on the overlap could still be improved. In this research, an overlapping percentage of 60 percent is used as a threshold. This threshold is not justified, but it could be justified in further research. Besides this, the method has the advantage that no further processing would be required to synchronize the objects in the databases.
- Research related to using the dates of the objects
  - Most BGT objects seem more up-to-date than the DTB objects. This is unlikely, because DTB objects are used to establish the BGT in the RWS data owner area. The threshold of 180 days is probably too low, because some objects such as buildings and roads have to be up-to-date within 6 months, but other objects have to be up-to-date with 18 months for instance. It could be investigated in further research if a higher threshold would improve the results. Another reason, which is not investigated, could be that it is caused by the first delivery in 2017. While the DTB objects are older and converted to BGT objects, the BGT objects got a new date and therefore they are more up-to-date.
  - It is not investigated if dates could be used as indicator without using data mappings, because correct dates could contain wrong mappings, and wrong dates could contain correct mappings. However, if there is no appropriate data mapping, it could be worth investigating.
- Other research
  - The method could be further automated, made more efficient and cleaner.
  - Literature about the harmonization of spatial data could be added. An example is literature on the website of the Humboldt project.

– **Most importantly**, the results must be compared with the ground truth to determine the accuracy of the synchronization. The ground truth refers to the real objects on the ground.

8. Reflection

I learned that there is a difference between the reality and the method in theory. To bring them closer together, I think that organisational management, plans, monitors and laws for geographic information are important.

Besides this, I liked to have freedom to develop my own method without considering all the literature in the field. I have searched for literature, but at first I could not find it. That is why I immediately started to implement my own method, which is mainly based on the RWS document [Wouters and Engelsma, 2019] and the acquired knowledge from Geomatics at the TU. I learned about the data layers of the BGT and DTB along the way, which was nice, but also stressful, because I got a lot of errors, that had to be solved.

If I had time left for the internship, I would have liked to add a comparison between my implemented method and the related literature about the harmonization of spatial data from the Humboldt project. I am not sure if I regret not knowing about the literature earlier, because sometimes literature limits my creativity and comparing the methods afterwards seems much more interesting to me. I do not mind to reinvent the wheel. Furthermore, I would create a ground truth to test the accuracy of the method.

The knowledge of many Geomatics courses such as Geo Database Management Systems, Python Programming for Geomatics and Geographical Information System and Cartography were used in the project. During the lectures, the BGT, conversions between data and references between objects were already discussed, which made the project much easier for me. My thesis requires knowledge about databases, which was useful for the project too.

A limitation of the project could be that I learned not many new things, because it was mainly based on the knowledge that I already had.

A. Appendix

A.1. MMS notification

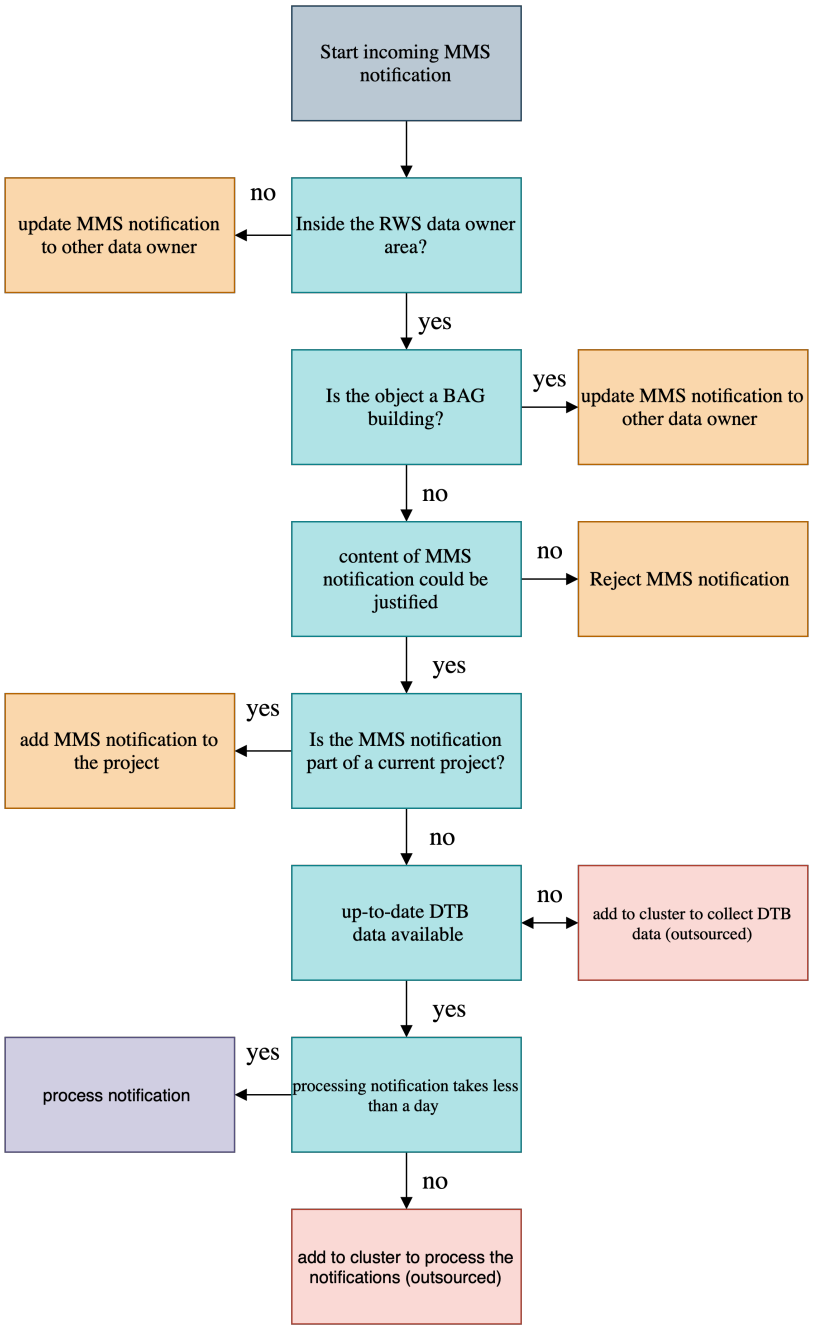


Figure A.1.: MMS

A.2. DTB quality

BGT-object	Type	Actualiteit van object in maanden	Positionele nauwkeurigheid tussen punten van object in cm	Idealisatie per punt van object in cm
Wegdeel		6	30	2 – 5
Onderst. wegdeel		18	30	5 - 10
Spoor		18	30	2 – 5
Onbegr. terreindeel		18	60	≥ 10
Begroeid terreindeel		18	60	≥ 10
Waterdeel		18	60	≥ 10
Ondersteunend waterdeel		18	60	≥ 10
Pand		6	30	0 – 2
Overig bouwwerk	Bassin	18	60	≥ 10
	Bezinkbak	18	30	2 – 5
	Lage trafo	18	30	0 – 2
	Open loods	18	30	0 – 2
	Opslagtank	18	30	2 – 5
	Overkapping	18	30	0 - 2
Overbruggingsdeel		6	30	0 – 2
Tunneldeel		6	30	0 – 2
Overig Kunstwerkdeel		18	30	0 – 2
Scheiding	Muur	18	30	0 – 2
	Kademuur	18	30	0 – 2
	Geluidsscherm	18	30	5 – 10
	Damwand	18	30	5 - 10
	Walbescherming	18	60	≥ 10
	Hek	18	60	≥ 10
Ongeclassificeerd object				
Functioneel gebied	Kering	18	60	≥ 10

Figure A.2.: The required objects of the BGT with the required quality [van Rossem, 2012]

A.3. database creation

```
import psycopg2
from psycopg2.extensions import ISOLATION_LEVEL_AUTOCOMMIT
from subprocess import call, run, PIPE
import os

def create_database():
    #1. Connect to database
    conn = psycopg2.connect("user=postgres password=1234")
    cur = conn.cursor()
    conn.set_isolation_level(ISOLATION_LEVEL_AUTOCOMMIT)

    #2. Create database
    name_database = "dtbdatabase"
    cur.execute("SELECT datname FROM pg_database;")
    list_database = cur.fetchall()
    if (str(name_database), ) in list_database:
        drop_database = "DROP DATABASE "+ name_database
        cur.execute(drop_database)
        conn.commit()
    create_database = "CREATE DATABASE "+ name_database
    cur.execute(create_database)
    conn.commit()
    conn.close()

    #3. Create extensions
    conn = psycopg2.connect("dbname=dtbdatabase user=postgres password=1234")
    cur = conn.cursor()
    create_postgis = "CREATE EXTENSION IF NOT EXISTS POSTGIS"
    cur.execute(create_postgis)
    conn.commit()

    create_sfcgal = "CREATE EXTENSION IF NOT EXISTS POSTGIS_SFCGAL"
    cur.execute(create_sfcgal)
    conn.commit()
    conn.close()
```

Figure A.3.: creation of databases with postgis and sfcgal

A.4. data mapping function

```
import psycopg2
import xlrd
import networkx as nx
import os

# AANPASSEN: linestring
# BGT: 'LINESTRING(139678.954 465524.3, 139678.954 469160.693, 142703.805 469160.693, 142703.805 465524.3, 139678.954 465524.3)'
# BGT2: 'LINESTRING(23082 398958, 23082 412612, 50654 412612, 50654 398958, 23082 398958)'
# BGT3: 'LINESTRING(95587 433303, 95587 436371, 100434 436371, 100434 433303, 95587 433303)'

def mapping_dtb(layer_name):
    #1. Import the excel file, the LookupTable
    wb = xlrd.open_workbook('/Users/karinstaring/Documents/rijkswaterstaat/implementation/data/BGTLookUp.xlsx')
    sheet = wb.sheet_by_index(0)
    num_rows = len(sheet.col_values(0))

    #2. Add columns to the DTB objects to insert BGT attributes
    vlakken = ["""ALTER TABLE {} ADD COLUMN IF NOT EXISTS dtb_geobgt character varying(80)""".format(layer_name),
               """ALTER TABLE {} ADD COLUMN IF NOT EXISTS dtb_plus character varying(80)""".format(layer_name),
               """ALTER TABLE {} ADD COLUMN IF NOT EXISTS dtb_fys character varying(80)""".format(layer_name),
               """ALTER TABLE {} ADD COLUMN IF NOT EXISTS dtb_func character varying(80)""".format(layer_name),
               """ALTER TABLE {} ADD COLUMN IF NOT EXISTS dtb_sub character varying(80)""".format(layer_name)]
    for add_column in vlakken:
        cur.execute(add_column)

    #3. Select the DTB data that intersects the area of interest (Linestring, reference system = 28992)
    query_vlakken = """select * from {} where st_intersects(shape, st_polygon('LINESTRING(139678.954 465524.3, 139678.954 469160.693, 142703.805 469160.
    cur.execute(query_vlakken)
    vlakken_in_area = cur.fetchall()

    k = 0
    obj_count = len(vlakken_in_area)

    #4. Iterate over all objects in the area of interest
    for dtb_object in vlakken_in_area:

        print("{:.2f}".format((k / obj_count) * 100), end='\r')
        k = k + 1

        rowList = []
        # A unique row has to be selected to the data mapping
        for i in range(num_rows):
            row = sheet.row_values(i)
            if dtb_object[17] == row[1]: # compare type
                if row[3] == '':
                    row[3] = None
                if dtb_object[18] == row[3]: # compare function
                    if dtb_object[3] == row[9]: # compare niveau
                        rowList.append(row)

        if len(rowList) == 0:
            print('length rowList is 0: ' + str(dtb_object))
        elif len(rowList) > 1:
            print('length rowList is more than 1: ' + str(dtb_object) + ' and the rowList: ' + str(rowList))
        else:
            # add BGT attributes to the DTB object
            sql = """ UPDATE {} SET dtb_geobgt = %s, dtb_plus = %s, dtb_fys = %s, dtb_func = %s, dtb_sub = %s WHERE dtb_id = %s""".format(layer_name)
            cur.execute(sql, (rowList[0][10], rowList[0][11], rowList[0][12], rowList[0][13], rowList[0][14], dtb_object[1]))
            conn.commit()
```

Figure A.4.: function `data_mapping`

A.5. delete and rename DTB attributes

```
import psycopg2
import xlrd
import networkx as nx
import os

columns = []
def mapping_dtb(layer_name, conn):
    #2. Drop/delete columns
    drop_attributes = [
        'pjt_id', 'bvk_id', 'gvk_id',
        'ivk_id', 'ovk_id', 'vvk_id',
        'wvk_id', 'dtm', 'attribuut_fout_ind',
        'coordinaat_fout_ind', 'muteerder', 'inwinner',
        'methode_inwinning', 'naam', 'dtb_mutatie_code',
        'ivri_mutatie_code', 'lengte', 'oppervlakte',
        'talud', 'shape_length', 'shape_area',
        'bkv_id', 'gdb_mutatie_code']

    for attribute in drop_attributes:
        drop_column = """ALTER TABLE {} DROP COLUMN IF EXISTS {}""".format(layer_name, attribute)
        cur.execute(drop_column)
        conn.commit()

    columns_in_layer = []
    layers = """SELECT column_name FROM information_schema.columns WHERE table_schema = 'public' AND table_name   = '{}'""".format(layer_name)
    cur.execute(layers)
    layers = cur.fetchall()
    for layer in layers:
        columns_in_layer.append(layer[0])
    print(columns_in_layer)

    #2. Alter/rename columns
    alter_attributes = [
        'dtb_id', 'niveau', 'datum_mutatie',
        'datum_inwinning',
        'type', 'functie', 'bronhouders',
        'shape', 'geobgt_dtb', 'plust_dtb',
        'fysiek_dtb', 'func_dtb', 'subt_dtb']

    renamed_attributes = [
        'dtb_id', 'dtb_niveau', 'dtb_datum1',
        'dtb_datum2',
        'dtb_type', 'dtb_oldf', 'dtb_bronh',
        'dtb_geom', 'dtb_geobgt', 'dtb_plus',
        'dtb_fys', 'dtb_func', 'dtb_sub']

    for i, attribute in enumerate(alter_attributes, 0):
        if attribute not in columns_in_layer:
            continue
        elif attribute == renamed_attributes[i]:
            continue
        else:
            alter_column = """ALTER TABLE {} RENAME COLUMN {} TO {}""".format(layer_name, attribute, renamed_attributes[i])
            cur.execute(alter_column)
            conn.commit()
```

Figure A.5.: function `data_mapping` to delete and rename DTB attributes

A.6. The function `union_geometries`

```
def union_geometries():
    #1. select attributes
    union_vlakken = """select bgt_id, dtb_id, bgt_bronh, layer, dtb_geobgt, bgt
    cur.execute(union_vlakken)
    union = cur.fetchall()

    #2. add category column
    add_compared = """ALTER TABLE public.union ADD COLUMN IF NOT EXISTS categor
    cur.execute(add_compared)

    k_union = 0
    union_count = len(union)

    #2. iterate over all objects
    for obj in union:

        print("{:.2f}".format((k_union / union_count) * 100), end='\r')
        k_union = k_union + 1
        #3. compare the attributes for every object

        attrList = [obj[0], obj[1], obj[2], obj[3], obj[4], obj[5], obj[6], obj
        category_value = compare_attributes(attrList)

        #4. update category column for the object
        sql = """ UPDATE public.union SET category = %s WHERE ogc_fid = %s"""
        cur.execute(sql, (category_value, obj[13]))
        conn.commit()
```

Figure A.6.: The function `union_geometries` calls the function `compare_attributes`

A.7. The function compare_attributes

```
def compare_attributes(obj):
    # Is data available [0, 1 and 2]
    further_testing = 'no'
    if obj[0] == None:
        if obj[1] == None:
            category_value = 'no_bgt_dtb'
        else:
            category_value = 'no_bgt'
    else:
        if obj[1] == None:
            if obj[2] == 'L0002':
                category_value = 'rws_no_dtb'
            else:
                category_value = 'no_dtb'
        else:
            category_value = 'dtb_bgt'
            further_testing = 'yes'

    # Is geobgt available? [3 and 4]
    if further_testing == 'yes':
        if obj[3] == None:
            further_testing = 'no'
            category_value = 'dtb_bgt_no_layer'
        if obj[4] == None:
            further_testing = 'no'
            category_value = 'dtb_bgt_no_geobgt'

    # Is geobgt the same? [3 and 4]
    if further_testing == 'yes':
        if obj[3].replace('bgt_', '').lower() == obj[4].replace('_v', '').lower():
            category_value = 'same_geobgt'
            further_testing == 'yes'
        else:
            category_value = 'different'
            further_testing = 'no'
```

Figure A.7.: The function compare_attributes part 1

```
# Is fysiekvoorkomen the same? [5, 6, 7 and 8]
if further_testing == 'yes':
    fys1_values = ['oever, slootkant', 'watervlakte', 'landhoofd', 'waardeOnbekend', 'transitie']
    fys2_values = ['transitie']
    fys3_values = ['transitie']
    if obj[5] != None:
        if obj[5] in fys1_values:
            category_value = 'onbekende_waarden'
            further_testing = 'no'
        elif obj[5] != obj[8]:
            category_value = 'different'
            further_testing = 'no'
        else:
            category_value = 'same_fysiekvoorkomen'
            further_testing = 'yes'
    elif obj[6] != None:
        if obj[6] in fys2_values:
            category_value = 'onbekende_waarden'
            further_testing = 'no'
        elif obj[6] != obj[8]:
            category_value = 'different'
            further_testing = 'no'
        else:
            category_value = 'same_fysiekvoorkomen'
            further_testing = 'yes'
    elif obj[7] != None:
        if obj[7] in fys3_values:
            category_value = 'onbekende_waarden'
            further_testing = 'no'
        elif obj[7] != obj[8]:
            category_value = 'different'
            further_testing = 'no'
        else:
            category_value = 'same_fysiekvoorkomen'
            further_testing = 'yes'
    else:
        if obj[8] == None:
            category_value = 'same_fysiekvoorkomen'
            further_testing = 'yes'
        else:
            category_value = 'different'
            further_testing = 'no'

# Is plustype the same? [9 and 10]
if further_testing == 'yes':
    plus_values = ['transitie', 'niet-bgt']
    if obj[9] != None and obj[10] != None:
        if obj[9] in plus_values:
            category_value = 'onbekende_waarden'
            further_testing = 'no'

        elif obj[9] == obj[10]:
            category_value = 'same_plus'
            further_testing = 'yes'
        else:
            category_value = 'different'
            further_testing = 'no'
```

Figure A.8.: The function `compare_attributes` part 2

A.8. The function `compare_dates`

```
    elif obj[9] == None and obj[10] == None:
        category_value = 'same_plus'
        further_testing ='yes'
    else:
        category_value = 'different'
        further_testing = 'no'

# Is function the same? [11 and 12]
if further_testing == 'yes':
    func_values = ['ruiterpad', 'voetgangersgebied', 'overweg', 'transitie', 'niet-bgt']
    if obj[11] != None and obj[12] != None:
        if obj[11] in func_values:
            category_value = 'onbekende_waarden'
        elif obj[11] == obj[12]:
            category_value = 'same'
        else:
            category_value = 'different'
    elif obj[11] == None and obj[12] == None:
        category_value = 'same'
    else:
        category_value = 'different'
return category_value
```

Figure A.9.: The function compare_attributes part 3

```python
def compare_dates(obj):
    layers_used_for_comparison = ['dtb_bgt_no_layer', 'dtb_bgt_no_geobgt', 'different']
    if obj[0] in layers_used_for_comparison:
        # select lv_formal
        if obj[2] != None and obj[3] != None:
            # convert dates
            lv_unfinished = obj[2].split('T')[0]
            lv = lv_unfinished.split('-')
            formal_unfinished = obj[3].split('T')[0]
            formal = formal_unfinished.split('-')
            lv_date = datetime.date(int(lv[0]), int(lv[1]), int(lv[2]))
            formal_date = datetime.date(int(formal[0]), int(formal[1]), int(formal[2]))
            # formal_date = more accurate
            if lv_date < formal_date:
                lv_formal = formal_date
            # lv_date = more accurate
            elif lv_date > formal_date:
                lv_formal = lv_date
            else:
                lv_formal = lv_date
        elif obj[2] != None:
            lv_formal = obj[2]
        elif obj[3] != None:
            lv_formal = obj[3]
        else:
            synchronization_value = 'not_possible'

        # checker presence of bgt_reality [1], dtb_collection [5] and dtb_mutation [4]
        if obj[1] == None or obj[4] == None or obj[5] == None:
            synchronization_value = 'not_possible'
        else:
            # convert dates
            bgt_datum1 = obj[1].split('-')
            bgt_reality = datetime.date(int(bgt_datum1[0]), int(bgt_datum1[1]), int(bgt_datum1[2]))
            if lv_date < bgt_reality:
                synchronization_value = 'not_possible'

            else:
                dtb_mutation = obj[4]
                dtb_collection = obj[5]
                if dtb_mutation < dtb_collection:
                    synchronization_value = 'not_possible'
                else:
                    cdiff = bgt_reality - dtb_collection
                    collection_diff = abs(cdiff.days)
                    # same object
                    if bgt_reality == dtb_collection or collection_diff < 180:
                        mdiff = lv_formal - dtb_mutation
                        mutation_diff = abs(mdiff.days)
                        # same mutation
                        if lv_formal == dtb_mutation or mutation_diff < 180:
                            synchronization_value = 'same_object_same_mutation'
                        # different mutation
                        else:
                            if lv_formal < dtb_mutation:
                                synchronization_value = 'dtb_mutation_more_accurate'
                            else:
                                synchronization_value = 'bgt_mutation_more accurate'
                    # different object
                    else:
                        if bgt_reality < dtb_collection:
                            synchronization_value = 'dtb_object_more_accurate'
                        else:
                            synchronization_value = 'bgt_object_more_accurate'

    # untill here: layers_used_for_comparison
    elif obj[0] == 'rws_no_dtb':
        synchronization_value = 'rws_no_dtb'
    elif obj[0] == 'same_func':
        synchronization_value = 'same_attributes'
    else:
        synchronization_value = 'None'
    return synchronization_value
```

Figure A.10.: The function `compare_dates`

Bibliography

RWS (2020a). Bgt productspecificaties 23 maart 2020.pdf.

RWS (2020b). Handboek dtb-droog en dtb-nat.

van Rossem, R. (2012). Basisregistratie grootschalige topografie: Managementsamenvatting bgt—imgeo standaarden. Technical report, Ministerie van Infrastructuur en Milieu.

Wouters, P. and Engelsma, E. (2019). Functioneel ontwerp bgt v0.2.