Open Geospatial Data, Software and Standards

ORIGINAL ARTICLE Open Access

Automatic conversion of OSM data into LULC maps: comparing FOSS4G based approaches towards an enhanced performance

J. Patriarca[1*], C. C. Fonte[1,2], J. Estima[3,5], J.-P. de Almeida[1,2] and A. Cardoso[2,4]

Abstract

OSM2LULC is a software package developed to automatically convert OpenStreetMap (OSM) data into Land Use Land Cover (LULC) maps using Free and Open Source Software for Geospatial (FOSS4G) tools. It needs to be highly efficient given the increasing detail of OSM data and the need to apply it to large extent regions. In this article, a comparison between the implementation of OSM2LULC in different available GIS platforms is made using both vector and raster data structures, which resulted in different versions. A description of the differences of each version is made and, to assess their performance, they were applied to four different study areas with different characteristics, in terms of available OSM data and area size. The performance of each version was evaluated taking into account: the overall processing time required to obtain LULC maps; and differences in the results obtained when different data structures (vector and raster) were used. Results showed that the adoption of a strategy that favors interoperability between FOSS4G and the combined use of both vector and raster data promotes a performance increase. After analysing the topological relationships of OSM data, the conversion to raster data format and the execution of procedural parts with such data indicated significant performance gains, without any positional distortions that significantly compromise the applicability of the final result in further case scenarios.

Keywords: GIS performance, FOSS4G, Interoperability, OSM to LULC

Introduction

Geospatial distribution of land-use and land-cover (LULC) is of most interest in both research and urban/ regional planning. LULC is typically one of the input parameters of models applied to several domains, e.g. environmental monitoring, multi-hazard risk & disaster modelling, urban & regional planning, natural resources management, or natural species distribution modelling [2, 4, 6, 9, 11, 16, 17, 28, 30, 34]. As such, efforts and resources spent in creating LULC maps are well justified by official mapping agencies as well as scientific researchers who constantly seek higher detail and quality at lower cost [8–10, 14, 29–31, 35].

Although traditional approaches to generate LULC data are usually based on aerial or satellite imagery classification, recent technology developments allow other options based on new data acquisition methods. In particular, developments on citizen-science, geo-crowdsourcing, and Web 2.0 based applications enable the creation, dissemination and update of volunteered geographic information (VGI). These allow the development of new approaches envisaging the derivation of LULC maps or their validation (e.g. [1, 3, 5, 7]). Regarding VGI resources of geospatial data, OpenStreetMap (OSM)[1] is one of the very first collaborative projects to provide free geospatial data. Because OSM covers a wide range of thematic domains across the whole world, it has a considerable potential for LULC mapping. Moreover, the fact that OSM data are continuously updated by its huge community of

* Correspondence: joaquimaspatriarca@gmail.com
[1]Institute for Systems Engineering & Computers at Coimbra (INESCC), Rua Sílvio Lima -Edifício DEEC, S.3.4, 3030-290 Coimbra, Portugal
Full list of author information is available at the end of the article

[1]https://www.openstreetmap.org

collaborators, makes it the largest, most diverse, complete, and up-to-date geospatial database [22]. As a key component in any derived data product, quality evaluation is even more important for VGI, given their volunteered nature. Several approaches are used to validate OSM data, which include validation by the crowd itself (e.g. volunteers can edit contributions and correct possible wrong data, apply algorithms to detect wrong edits created, for example, by vandalism), as well as validation by experts [13, 20, 21, 25]. Tools to assist contributors in the creation of high quality data are also available, such as those to correct feature shapes (for example, to convert irregular polygons into a rectangular shape), or the availability of snapping mechanisms for line connection purposes.

The overall need of LULC maps along with the availability of a large amount of free geospatial data in OSM became the motivation for the development of methodologies to generate LULC information from OSM. Arsanjani et al. [3] tested the conversion of OSM data into a LULC map using the Urban Atlas nomenclature with encouraging results. Fonte et al. [8, 10] proposed an automated methodology to convert OSM data into LULC maps and Fonte et al. [9] showed the potential of merging LULC data extracted from OSM with existing LULC products, namely the GlobeLand30. The use of OSM extracted LULC data to validate LULC maps was also tested with promising results, as for Urban Atlas level 1 nomenclature the accuracy indices obtained using OSM data (where available) to obtain the reference class were not very different from the ones obtained when the reference class was always obtained by photo interpretation [7]. Shultz et al. [29] produced a global Land Cover product using OSM data, and used the available data to train a classifier that was used to classify satellite imagery in order to generate data for the regions when OSM data is not available. The encouraging results obtained in the previous efforts motivated the creation of OSM2LULC – a FOSS4G aiming to automatically convert OSM data into LULC maps [8–10]. OSM2LULC[2] is part of the GeoData Algorithms for Spatial Problems (GASP) Python package,[3] whose ultimate aim is to provide various tools to extract, to convert, to analyse, and to validate geospatial information. Furthermore, OSM2LULC is free software under GNU-GPL v3.0 [12, 15, 18, 19, 27, 32, 33]. The decision of making OSM2LULC a free software has been made to provide a tool without usage restrictions to researchers and GIS analysts who need to work with LULC information.

The decision to make OSM2LULC free software unrestricted has been made to make free of charge an important tool for GIS researchers and analysts who need to work with LULC information.

OSM2LULC constitutes a sequential logical integration of several tools belonging to other FOSS4G packages. Such integration establishes a relation between the OSM features and the LULC classes of the LULC nomenclatures of Urban Atlas, Corine Land Cover, or GlobeLand30. In cases where such relation is not direct, OSM2LULC uses other geographic information system (GIS) software packages to analyze their geometric and topological properties to judge whether or not they should be associated with a certain LULC class. For line-based OSM features, these geometries are converted into polygons by creating buffer zones using the distance (predefined or calculated with spatial analysis approaches) to other OSM features. Finally, after associating each OSM feature to a LULC class, inconsistencies are frequent and became evident when there are overlapping regions classified with different classes. The algorithm eliminates remaining inconsistencies (or instance, when there are overlapping regions classified with different classes) by applying a rule based on different priorities assigned to each LULC class (see Table 1 for an example), which aggregates all classes in a single vector or raster based theme [9].

OSM2LULC has already been applied to different case study regions [8–10]. These case studies allowed the identification of the main problems of the algorithm. First, the results quality was identified as a key problem hampering the use of OSM2LULC due to: 1) quality of the original OSM data; 2) lack of additional rules to assign OSM features to the correct LULC class, which should be based on the 2D and 3D topological relations between OSM features. In addition, the lack of performance was also identified as a key problem capable enough of decreasing the interest in using OSM2LULC. Increasing efficiency is a key aspect in future developments of OSM2LULC, since it will allow the production of detailed LULC maps for very large extent areas

Table 1 Example of priorities referring to level 2 of Urban Atlas (UA)

Priority	UA Level 2 Classes	Class name
1	1.2	Artificial surfaces
2	5.0	Water
3	1.4	Artificial non-agricultural vegetated areas
4	1.3	Mine, dump and construction sites
5	1.1	Urban Fabric
6	2.0	Agricultural, semi-natural areas, wetlands
7	3.0	Forests

[2]Available at https://github.com/jasp382/gasp/tree/master/gasp/osm2lulc
[3]Available at https://github.com/jasp382/gasp

(countries or continents) and the regular derivation of LULC maps from OSM data to map changes in LULC distribution, taking advantage of the fact that OSM data is constantly being updated. Furthermore, performance increment may be essential to maximize the number of OSM2LULC users: 1) motivation for GIS programmers; 2) possibility of making the derivation of LULC from OSM data available on the Internet as a web service, so that non-programmers can also use it.

Moving on that direction, this work aims to compare various implementation approaches for OSM2LULC and quantify their performance. To verify to which extent these could be generalized, we tested and evaluated such implementations over four study areas with different characteristics. The ultimate aim was to identify and minimize performance issues and to evaluate the impact of such strategies on the final product.

The remainder of the article is structured as follows: section 2 describes the characteristics of OSM data and the study areas, and describes the methodology followed. Section 3 describes the implementation of OSM2LULC, along with the several versions that were implemented. Section 4 discusses the results and section 5 draws some conclusions and presents ideas for future work.

Data and methodology

Data

OSM data is formed by the following elements: points (called nodes), ways (which may be open or closed lines, or polygons), and relations (which may be used to connect elements that are not physically connected).[4] Tags can be associated with the elements, which are formed by a key and a value - for example: "building" = "school", where "building" is the key and "school" is the value. Each element may have one or multiple tags. For example, a line representing a street may have the tag "highway" = "secondary" and an additional tag with the name of the street "name" = "Street A". There is a long list of tags suggested by the OSM community,[5] but mappers can create additional tags. This, on one hand has the advantage of not imposing restrictions to the volunteers edits, which contributes to the richness of OSM, but on the other hand makes the use of the existing OSM data more complex, as, for example, different tags may be found in different regions, the meaning of some tags may not be clear to OSM users, and different tags may be used to represent the same type of features.

The current implementation of OSM2LULC considers only the tags listed in the OSM Map Features wiki page.[6] Table 2 shows some of the most important tags providing data about LULC used in the conversion.

This work uses four study areas to test the implementation methodologies described in section 2.2, located in the regions of Coimbra (Portugal), Lisbon (Portugal), London (United Kingdom) and Milan (Italy). Table 3 presents their characteristics and Fig. 1 a) - d) show the OSM data available for them.

All tests performed in this work may be replicated using osm2lulc.ipynb, which is a jupyter notebook available at GitHub, as part of the GASP Python package.[7] To use this notebook, the GASP Python package must be configured correctly, according the instructions provided in the GASP GitHub page (https://github.com/jasp382/gasp, accessed July 5, 2019).

Methodology

To achieve the objectives defined in the previous section, the methodology proposed in this work is separated into three distinct parts.

The first part covers the restructuring and optimization of the code from the first version of OSM2LULC (Version 1.0), described in [8–10]), which resulted in a new version (Version 1.1). Two essential aspects were changed and improved:

1. The code structure was modified - in version 1.0, the processing was based on LULC classes, that is, for each LULC class there was an associate module/sub-procedure; whereas Version 1.1 has been organized according to the type of operations/calculations performed over the data - i.e. one module for each single sequence of tasks. Thus, processing of LULC classes involving the same conversion strategies is now performed within the same OSM2LULC module. This approach eliminated sequences of tasks that were replicated throughout the source code, accelerating, in turn, the production of the final LULC map;
2. GDAL replaced *osm2pgsql*[8] and *osmosis*[9] and is now used to convert OSM XML files into files readable by any GIS software.

[4]https://wiki.openstreetmap.org/wiki/Beginners_Guide_1.3, accessed March 10, 2019
[5]https://wiki.openstreetmap.org/wiki/Map_Features, accessed March 10, 2019
[6]https://wiki.openstreetmap.org/wiki/Map_Features, accessed March 10, 2019
[7]osm2lulc.ipynb is available at https://github.com/jasp382/gasp/blob/master/gasp/osm2lulc/osm2lulc.ipynb (accessed July 5, 2019).
[8]https://github.com/openstreetmap/osm2pgsql, accessed March 10, 2019.
[9]https://github.com/openstreetmap/osmosis, accessed March 10, 2019.

Table 2 List of some OSM tags (keys and values) used in the conversion to LULC classes

Key	Key values
amenity (area)	animal_shelter, arts_centre, bank, bar, brothel, cafe, car_rental, car_wash, casino, cinema, clinic, college, community_centre, courthouse, crematorium, crypt, dentist, dive_centre, driving_school, embassy, fast_food, ferry_terminal, fire_station, food_court, fuel, grave_yard, gym, hospital, internet_cafe
amenity (point)	bar, hospital, pub, restaurant, school, university
building (area)	apartments, cathedral, chapel, church, civic, commercial, garage, hangar, hospital, hotel, house, industrial, kiosk, mosque, office, public, residential, retail, school, shrine, stadium, synagogue, temple, train_station, transportation, warehouse, yes
building (point)	hotel, office, commercial,hospital, industrial, retail, warehouse, cathedral, chapel, church, mosque, temple, synagogue, school, stadium, train_station, transportation, public, shrine, civic, hangar, kiosk
highway (line)	living_street, motorway, motorway_link, pedestrian, primary, primary_link, residential, road, secondary, secondary_link, service, tertiary, tertiary_link, trunk, trunk_link, unclassified
landuse (area)	allotments, beer-garden, brownfield, cemetery, commercial, construction, depot, farm, farmland, farmyard, flowers, forest, grass, greenfield, greenhouse-horticulture, industrial, landfill, meadow, military, orchard, plants, pond, quarry, railway, recreation_ground, recreational_area, recreational, reservoir, residential, retail, scrub, village_green, vineyard
leisure (area)	adult_gaming_centre, amusement_arcade, beach_resort, dance, dog_park, garden, golf_course, hackerspace, ice_rink, marina, miniature_golf, nature_reserve, park, pitch, playground, sports_centre, stadium, summer_camp, swimming_area, swimming_pool, track, track, water_park
natural (area)	bay, bare_rock, beach, fell, forest, glacier, grassland, heath, mud, park, sand, scree, scrub, shingle, water, wetland, wood, riverbank
railway (line)	rail
waterway (line)	dock, river, riverbank, stream

Version 1.1 of OSM2LULC was defined as the baseline implementation for the discussion developed in the next sections.

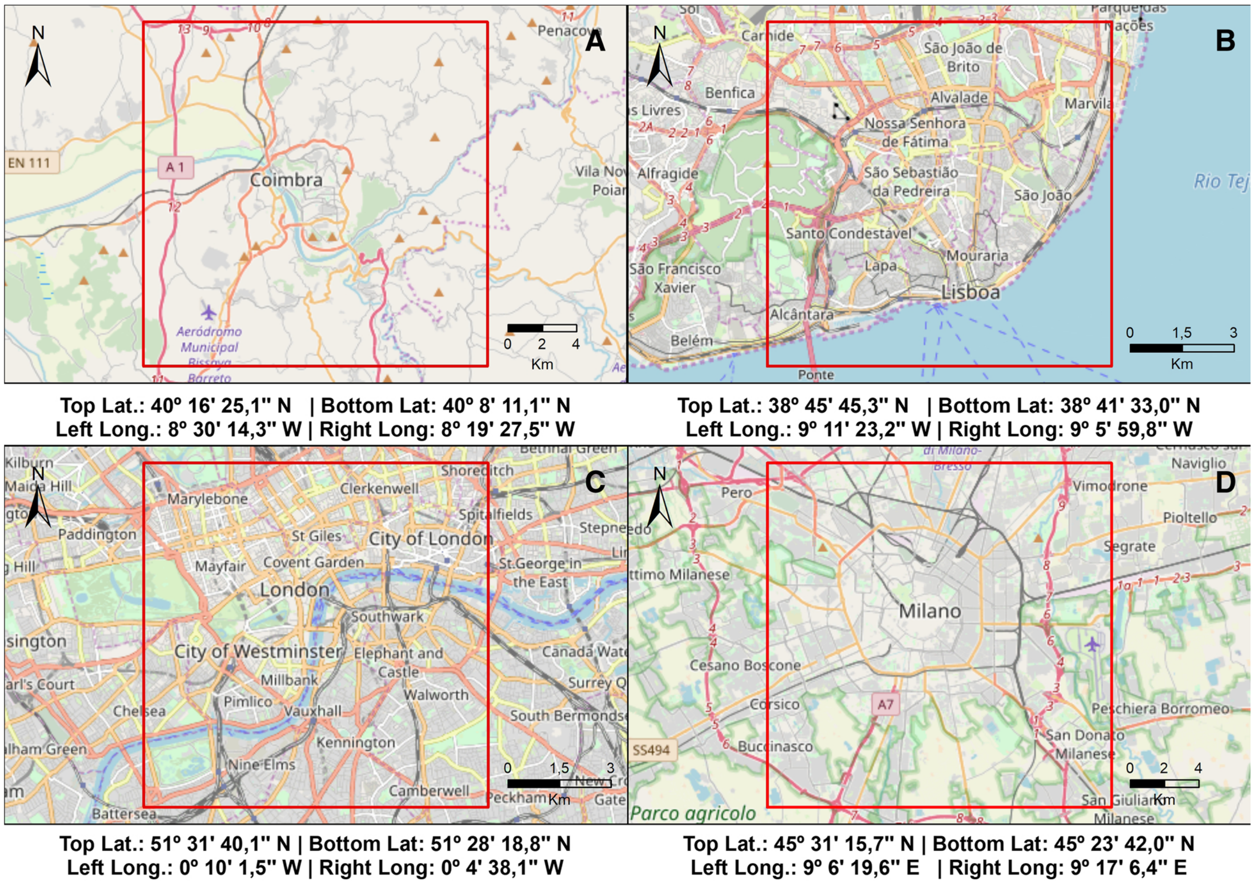
The second part referred to testing if the processing time of Version 1.1 could be improved. To achieve this, three more implementations were developed and tested: Version 1.2, 1.3 and 1.4.

The third part was related to the application of the four aforementioned implementations (Versions 1.1, 1.2, 1.3 and 1.4) to OSM data, covering different geo-spatial locations with different volumes of OSM data. Different study areas, indicated in section 2.1 (Table 3 and Fig. 1), were used in order to understand two types of differences:

1. Differences in terms of the program overall execution time, as well as values of the execution time of particular tasks. Those differences enable two types of analysis: on one hand, comparing results obtained for the same case study, it is possible to understand which version obtains better results in terms of performance time; on the other hand, comparing the results obtained for different case studies gives an idea of the capacity that each OSM2LULC version has for processing different volumes of data, and the impact of data volume on the application performance. In cases where a raster data model was used, additional analyses were accomplished using several cell dimensions (10, 5, and 2 m respectively), allowing an understanding to what extent different cell size conditions affect the overall algorithm performance;
2. Differences resulting from the usage of several OSM2LULC implementations which may in fact generate different results (as explained below in Section 4). The identification and quantification of areas classified differently by each implementation were undertaken through a process applied individually to results obtained for each study area (which is outlined in Fig. 2). This procedure consists of: identifying all possible pairs of results; considering each of these pairs individually, the

Table 3 - Characteristics of the study areas

Characteristics	Study areas			
	Coimbra (Portugal)	Lisbon (Portugal)	London (United Kingdom)	Milan (Italy)
Area (km^2)	400	100	100	400
Size of OSM data available (MB)	26	60	109	196
Number of OSM features to be processed	19,449	66,696	128,410	235,887
Number of OSM features used in more demanding operations	11,425	36,711	29,964	67,176



Top Lat.: 40º 16' 25,1'' N | Bottom Lat: 40º 8' 11,1'' N
Left Long.: 8º 30' 14,3'' W | Right Long: 8º 19' 27,5'' W

Top Lat.: 38º 45' 45,3'' N | Bottom Lat: 38º 41' 33,0'' N
Left Long.: 9º 11' 23,2'' W | Right Long: 9º 5' 59,8'' W

Top Lat.: 51º 31' 40,1'' N | Bottom Lat: 51º 28' 18,8'' N
Left Long.: 0º 10' 1,5'' W | Right Long: 0º 4' 38,1'' W

Top Lat.: 45º 31' 15,7'' N | Bottom Lat: 45º 23' 42,0'' N
Left Long.: 9º 6' 19,6'' E | Right Long: 9º 17' 6,4'' E

Fig. 1 Study areas used to test the performance of implementation approaches: **a**) Coimbra, **b**) Lisbon, **c**) London, **d**) Milan

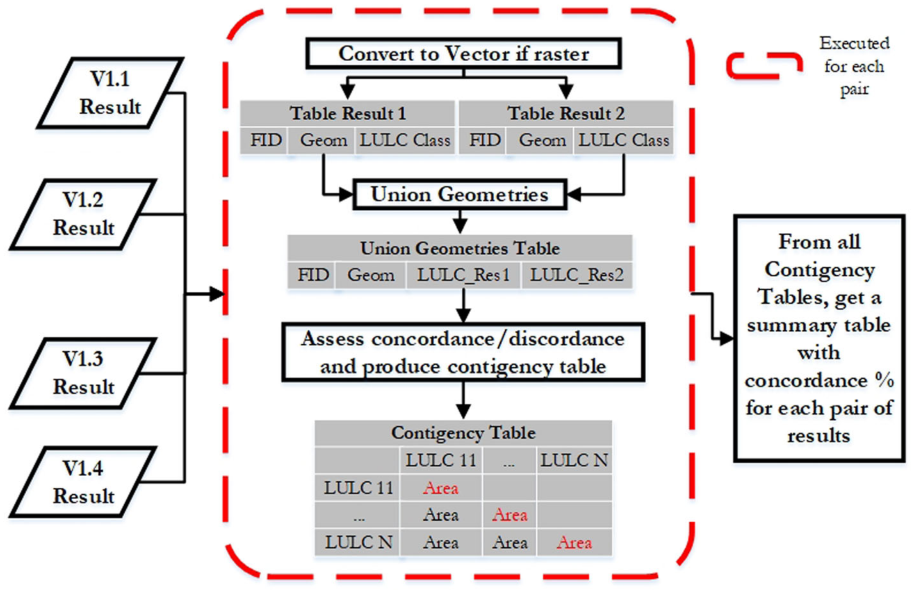


Fig. 2 Assessing agreement and discordance between results obtained by different implementations of OSM2LULC algorithm

geometric union of a pair was performed; in turn, the union result was then used to generate the contingency tables with the agreements and disagreements by LULC class; finally, a single table was created based on all contingency tables that systemize agreement/disagreement percentages of each pair of results.

Regarding the last aforementioned point, although the evaluation of the quality and thematic accuracy of the results generated by OSM2LULC would be an important topic, it is out of scope in this work. The assessment performed within this paper relates only to evaluating the impact of vector to raster conversions executed in some implementations, i.e. we are only assessing if the improvement we are having in terms of performance by running some implementations using raster formats results in major differences on the output.

All the performance benchmarks comparing the different implementations were performed on a computer with the following characteristics: Intel® Core™ i7–6700 CPU 3.40Ghz ×64-based; 16 GB of RAM; Solid State Driver with 1 TB; Ubuntu 18.04 64 bit LTS.

OSM2LULC implementation

Implementing support technologies

To minimize development efforts, OSM2LULC was implemented in Python programming language. All processes of the conversion algorithm can be implemented using functionalities and tools provided by various FOSS4G (e.g. GRASS GIS, SAGA GIS). Thus, it is possible to implement and run these tools from a wide variety of FOSS4G in the same script through their Python based API. The OSM2LULC development process was faster due to this choice (no need of developing spatial analysis tools from scratch) and allowed us to take best advantage of each package. Further to this orientation, it was necessary to decide which FOSS4G to be chosen and respective tools to be integrated. Such procedure was based on two steps, as follows: selection of most prominent alternatives based on literature review from Patriarca [26]; definition of a set of criteria to build a hierarchy of the pre-selected alternatives in the previous step.

Among a wide variety of FOSS4G packages, GRASS GIS,[10] gvSIG,[11] QGIS,[12] GDAL/OGR,[13] SAGA GIS,[14] and PostGIS[15] stand out from others for their maturity and robustness. In fact, with the exception of SAGA GIS, all these packages are "OSGeo projects", which proves its quality – one of the main goals of OSGeo foundation[16] is to ensure a high degree of quality of projects supported by them. The criteria considered to select the software used in the development of OSM2LULC are listed in Table 4. GRASS GIS 7.6.0 was selected since it fulfilled all criteria. In addition, GDAL / OGR 2.4.0 was also adopted for ensuring interoperability between OSM data and GRASS GIS (GRASS GIS can't read OSM files without GDAL/OGR).

After the selection process of the FOSS4G to be used in the development of OSM2LULC software package, its Versions 1.0 and 1.1 were implemented (Version 1.0 was then discarded and replaced by Version 1.1 - see Section 2).

Some performance tests of Version 1.1 revealed a few limitations. Seeking performance maximization, further versions were considered and developed. For the new versions, the FOSS4G packages compared in Table 4 were tested to find tools to replace the slower ones of Version 1.1.

Implementation structure

OSM2LULC software is based on a chain of logical procedures for the derivation of LULC maps from OSM data. All developed versions followed the logic outlined in Fig. 3; differences between them are only procedural relating: i) to the execution of the logical sequence of tasks inherent to assumptions of each module; ii) to the application of the hierarchical approach used to aggregate results of the various modules into a single layer. Versions 1.1 to 1.4 were developed upon a set of six modules prepared to address three needs, as follows:

- To generalize, to simplify, and to remove errors present in OSM feature geometries, namely in those cases where there are overlapping polygons that must be associated with same class;
- To determine whether the assignment of a LULC class to a particular OSM feature is the right one or is, at least, the more likely one, based on some kind of topological relationship between geometries or geometric properties;
- In the case of line-based OSM features, to transform this type of geometry into polygons according to a certain criterion. This effort is essential to ensure that information relating to roads and railways is not lost, since it is not included in the polygon-based layer of the OSM file. In addition, in a closer representation of reality, geographical entities represented in the line-based layer should be represented as polygons – recall that a road has a

[10]https://grass.osgeo.org/, accessed March 10, 2019.
[11]http://www.gvsig.com/en/home, accessed March 10, 2019.
[12]https://qgis.org/en/site/, accessed March 10, 2019.
[13]https://www.gdal.org/, accessed March 10, 2019.
[14]http://www.saga-gis.org/en/index.html, accessed March 10, 2019.
[15]http://postgis.net/, accessed March 10, 2019.
[16]https://www.osgeo.org/, accessed March 10, 2019.

Table 4 Criteria used to select which GIS Software to be integrated in OSM2LULC

	GRASS GIS	gvSIG	QGIS	GDAL/OGR	SAGA GIS	PostGIS
Do they provide fundamental tools for the envisaged processes?	Yes	No tool to compute near distance between features is available	Yes, but most of the relevant tools were from other FOSS4G	Yes, but depends of other packages (e.g. Spatialite) to accomplish some analysis	No tool to compute near distance between features is available	Yes
OSGeo project?	Yes	No, by the time our work started	Yes	Yes	No	Yes
Quality documentation available?	Yes	No	Yes	Yes	Yes	Yes
Examples of its application available?	Yes	Yes	Yes	Yes	Yes	Yes
Python-based API available?	Yes	No	Yes	Yes	Yes	Yes
Is the implementation of tools in Python Scripts easy?	Yes	No	Yes	No. All types of analysis could be done, but some would need additional configurations and the use of SQL (e. g. difference of polygons or near distance analysis)	Yes	No, layers are treated as database tables; advanced SQL queries are needed for some tools
Previous experience of the team?	Yes	No	No	Yes	No	Yes

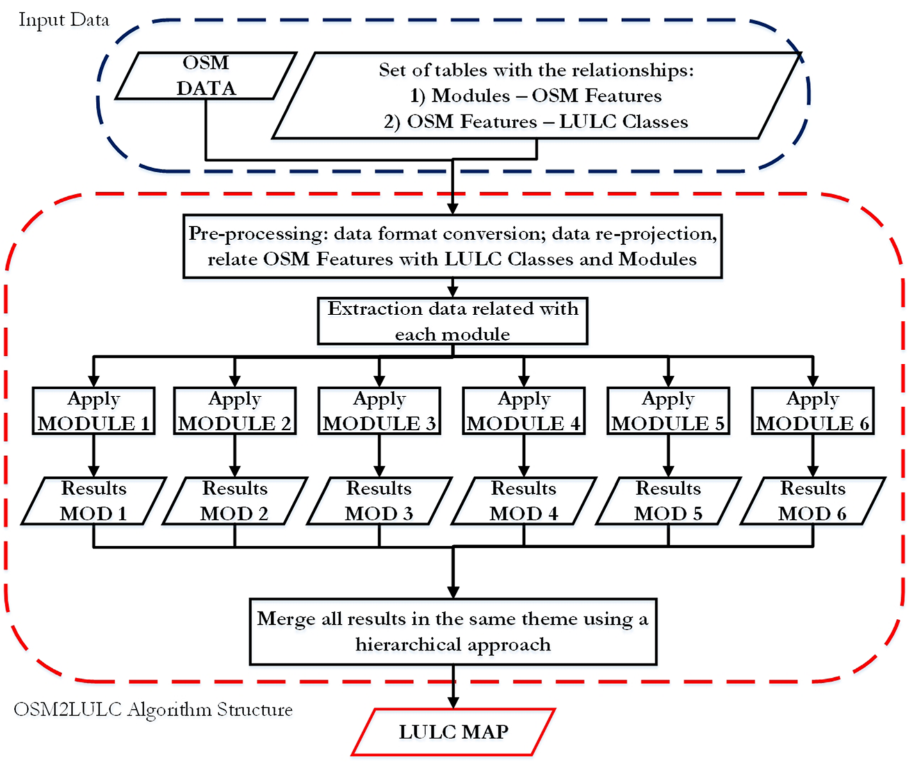

Input Data
OSM DATA
Set of tables with the relationships: 1) Modules – OSM Features 2) OSM Features – LULC Classes
Pre-processing: data format conversion; data re-projection, relate OSM Features with LULC Classes and Modules
Extraction data related with each module
Apply MODULE 1
Apply MODULE 2
Apply MODULE 3
Apply MODULE 4
Apply MODULE 5
Apply MODULE 6
Results MOD 1
Results MOD 2
Results MOD 3
Results MOD 4
Results MOD 5
Results MOD 6
Merge all results in the same theme using a hierarchical approach
OSM2LULC Algorithm Structure
LULC MAP

Fig. 3 Schema of OSM2LULC implementation

certain geospatial extent and thus has a certain area on the ground.

The implementation process was then based on the creation of six modules. Each module can be defined as a set of procedures that should be applied to a specific group of OSM features, transforming them into LULC classes, which are then merged together and ambiguities solved to generate the final LULC map (see Fig. 3).

The implementation requires a prior association of the various OSM features involved: 1) to LULC classes; and 2) to each module, depending on the type of processing required by each feature type, such as the creation of a buffer around the feature or its validation based on a set of rules (e.g. its area or neighbouring features). These relationships were declared in a relational database (see Fig. 4), hereinafter referred to as OSM2LULC Support DB, which is accessed each time the program runs. In some cases, the association between the various OSM features and the respective module is accompanied by other attributes (i.e. columns buffer_dist and area, as depicted in Fig. 4). Column area is used in situations where the conversion to LULC depends on the area (in square meters) of the OSM Polygon Features; column buffer_dist has the buffer distance used to transform lines into polygons (Module 2 and 5).

In subsection 3.3.1 the pre-processing phase necessary for the application of the logical steps performed in the modules is described and in section 3.3.2 each of the six modules is explained.

Pre-processing applied to OSM data

All versions of OSM2LULC apply the same set of preliminary tasks, which are intended to prepare the OSM data, consisting of:

- The original OSM file is transformed to: i) an SQLITE database for Version 1.1; or ii) a PostgreSQL database (PSQL DB) for Versions 1.2, 1.3 and 1.4;
- Based on the existing relationships in OSM2LULC Support DB, several queries are applied to update the original tables, adding new columns, which indicate: 1) the module that should be applied to each OSM feature; 2) LULC class (or classes) corresponding to a particular OSM feature; and 3) default values to be applied at specific processing times (i.e. fields buffer_dist and area, as shown in Fig. 4);
- Finally, the original spatial reference system (SRS) of OSM needs to be changed; because the original WGS84 (EPSG: 4326) is not projected; in fact, given that processing tasks involve operations comprising distance and area calculation, the various OSM2LULC modules have been prepared to receive projected SRS input data.

Modules description

Module 1 MODULE 1 is applied to OSM features that have a direct and unambiguous relationship with a LULC class. From a procedural point of view, this module only

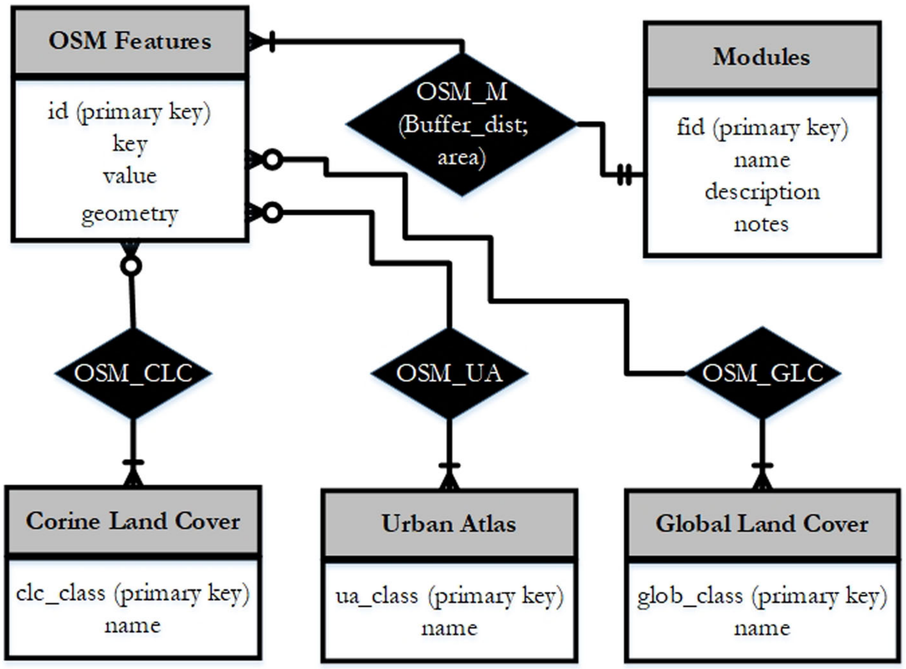


Fig. 4 Entity-relationship model for the database supporting the OSM2LULC Versions

presupposes the simplification and generalization of geometries associated with a given LULC class.

Module 2 MODULE 2 is used to transform line-based OSM features into polygons by creating "buffer zones", where the "buffer distance" depends on the distance between lines and buildings. This is the case of OSM Lines representing highways; this module is applied exclusively to this type of entities. The width of buffer zones associated with each line corresponds to the minimum distance between this feature and the nearest building multiplied by two. This rule was defined this way because the buffer distance for roads in urban areas should not exceed the minimum distance between the road axis and the nearest building. However, this rule is not applied when the distance between lines and nearest buildings is greater than 12 m, since it would be unlikely to find a highway with a width greater than 25 m in urban areas. In previous cases, a default buffer distance is applied according to the width that a highway with a certain "key/value" is expected to have.

MODULE 3 and MODULE 4 MODULE 3 and MODULE 4 are used in cases where the OSM feature-LULC Class relationship only occurs if the OSM feature's area is bigger (MODULE 3) or smaller (MODULE 4) than a default value (i.e. a threshold) predefined for that OSM feature, which depends on the OSM feature tag. For example, a feature with a "key" equal to "Landuse" and a "value" equal to "Forest" can be associated with more than one LULC category (forests or urban green spaces); the area of the feature's polygon is used to determine which LULC class it belongs to: if the area is bigger than 10,000 m2, the feature is treated as "Forest", otherwise it is considered as "Green Urban Space".

Module 5 MODULE 5 transforms line-based OSM features representing railways or waterways into polygon features by creating buffer zones around them. Unlike MODULE 2, which takes into account a comparison of distances, in MODULE 5 "buffer distance" is a parameter previously defined for each OSM feature.

Such strategy entails though some drawbacks. For instance, for different study areas, the area occupied by a body/watercourse feature may be quite distinct, so the definition of a static buffer distance is not the best option. In the future, developments are planned to eliminate this limitation, either through spatial analysis or through integration of other sources, namely satellite imagery.

Module 6 Finally, MODULE 6 complements the information associated with OSM polygons that have the value "yes" for the key "building" with complementary information from the point-based building representation, which was previously associated with LULC classes. This is achieved through the intersection of both geometries and allows to complete the information about the buildings and hence helps to reduce the uncertainty around the classification of the OSM features, as, for example, residential, industrial or commercial. When it is not possible to obtain further information about the buildings, they are assumed to be residential and will be associated with the corresponding LULC class.

It should be stressed that all modules above are independent of one another and there is no data sharing between them, i.e. the output of a certain module is never the input of another.

Priority rule

After applying the aforementioned modules, results produced by all modules are integrated. This is done by aggregating features corresponding to each LULC class into a single layer. Additionally, this process also seeks to tackle some inconsistencies that may still occur, such as cases of overlapping polygons that have been classified with distinct LULC classes. Such fact is very much related to the nature of OSM itself and may occur for different reasons: classification of OSM features with keys or values that do not characterize correctly the type of land-use or land-cover; lack of accuracy in defining geographical edges of entities; or simply because in reality entities above overlap or include one another – i.e. roads or railway lines over river lines; green spaces that are within urban areas [9]. In order to tackle these problems, a hierarchical approach was adopted consisting on the assignment of different levels of priority to LULC classes considered in the OSM data transformation process; such hierarchy of priority levels is based on the importance, size, spatial relationships, and topology of OSM data (see Table 1). As an example, Fig. 5 demonstrates how the hierarchical rule is applied using some classes in Table 1.

Differences among OSM2LULC versions

Version 1.1 were implemented using GRASS GIS. This version workflow is described in Table 5. Some performance tests of this code revealed some limitations to process datasets similar of those in Table 3 in a short time window. If for some reason, we seek to get results in less than 5 min, the definition of a zone of interest will be highly conditioned in terms of its geographic extent.

Aiming to minimize restrictions in applications of OSM2LULC, (e.g., making it available as a web service), performance improvement was sought based on the replacement of some tools in time consuming processes. Version 1.2 of OSM2LULC was the result of such

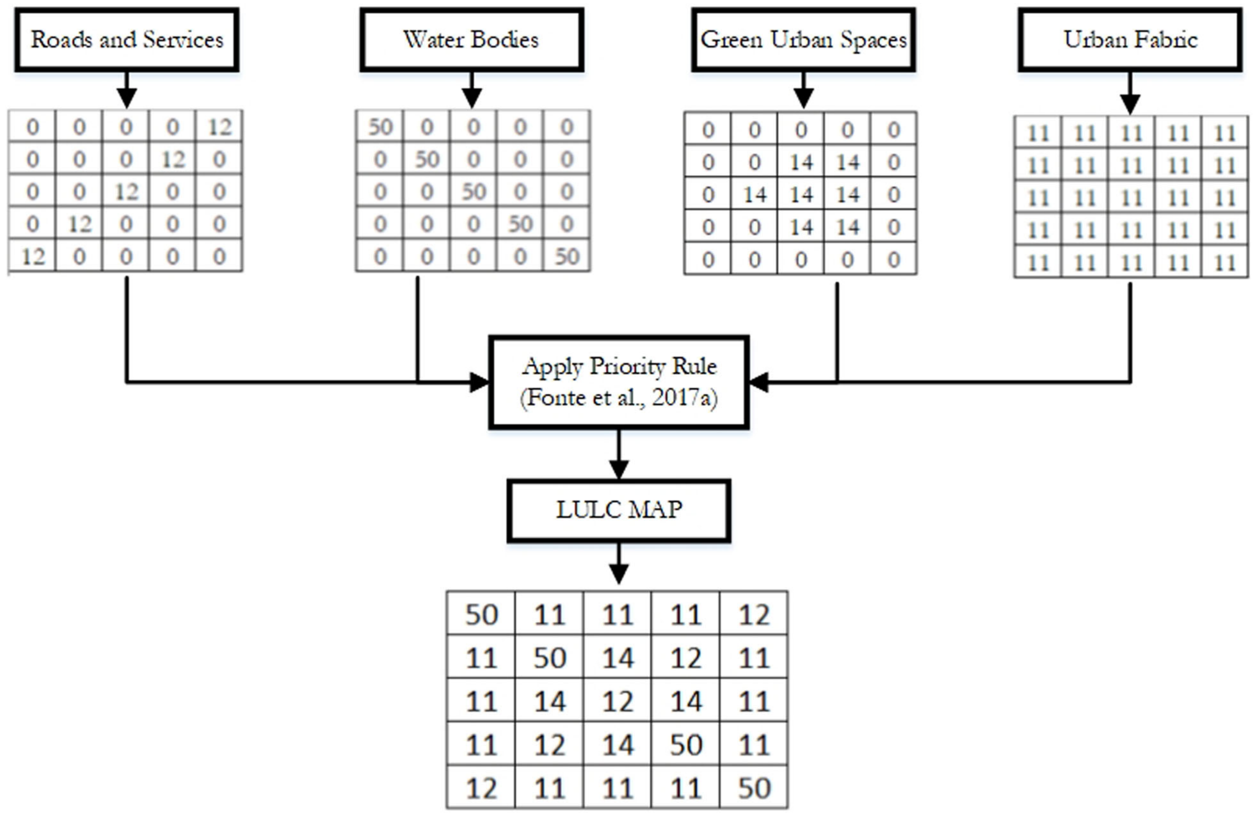


Fig. 5 – Priority rule exemplification

improvement - this implementation used the baseline approach defined in Table 6, but some of the spatial analysis operations were run inside a PostgreSQL database, using PostGIS tools (these operations are those in which GRASS GIS data processing is more time consuming).

Nevertheless, the general performance of the new version was not totally satisfactory, mainly due to limitations in the tool used in the priority rule application. Therefore, a new hypothesis was formulated based on two assumptions, as follows:

- Although limiting beforehand the quality of results, due to the simplification of a given geospatial geometry shape, better performance can be achieved using raster data format instead of vector format when applying the priority rule. Such change implies a simplification of geospatial analysis tasks for operations no longer dependent on the irregularity of vector geometries;
- In applying the priority rule, better performance may also be achieved by replacing the usage of FOSS4G tools for a Numpy-based[17] implementation of these operations.

[17] http://www.numpy.org/, accessed March 10, 2019

Versions 1.3 and 1.4 were then developed based upon the assumptions described above. Version 1.3 implemented only the first assumption, whereas version 1.4 implemented both together. In Version 1.3, at certain point vector data were converted to a raster data model within GRASS GIS and the priority rule operations were carried out using that data model. Area and distance calculations and buffering operations were performed on vector data. In the implementation of Version 1.4, OSM data was converted into Numpy arrays using GDAL 2.2.4 and the priority rule operations were accomplished using Numpy 1.15 tools. Area and distance calculations and buffering operations were done by PostGIS functions within a PostgreSQL database, avoiding GRASS GIS entirely (see Table 5).

Results and discussion

In this section we present the results of applying the aforementioned four versions of OSM2LULC to the different study areas described in section 2.1. Such tests provided us with results both related to their execution time and also enabled the comparison of the different LULC outputs resulting from the conversion operations.

Table 5 OSM2LULC Version 1.1 workflow and main differences in subsequent versions

	Version 1.1 - Baseline workflow/approach[a]	Differences in subsequent versions
MOD 1	Import data related with Module 1 into GRASS GIS (v.in.ogr) > Generalize data (v.dissolve)	Vers. 1.2 - Same as Version 1.1. Vers. 1.3 - Vector to raster (v.to.rast) replaced v.dissolve. Vers. 1.4 - GRASS GIS was set aside; data was exported to file (ogr2ogr) and converted to raster using GDAL Python Bindings.
MOD 2	Import data related with Module 2 (roads and buildings) into GRASS GIS (v.in.ogr) > Calculate distance between roads and buildings (v.distance) > calculate buffer using distances obtained or default thresholds (v.buffer) > dissolve polygons (v.dissolve).	Vers. 1.2 - Procedure is done inside PGSQL DB; results are imported to GRASS GIS at the end (v.in.ogr); ST_Distance and ST_DWithin replaced v.distance; ST_Buffer and ST_Unary Union replaced v.buffer and v.dissolve. Vers. 1.3 - One difference from Version 1.2: vector data is converted to raster (v.to.rast) at the end. Vers. 1.4 - Three differences from Version 1.2: GRASS GIS was set aside; ST_Unary Union was not used to dissolve buffer polygons; data was converted to raster using GDAL Python Bindings.
MOD 3 and 4	Select data related with Module 3/4 and bigger/smaller than threshold > Import it to GRASS GIS (v.in.ogr) > Generalize data (v.dissolve)	Vers. 1.2 - Same as Version 1.1. Vers. 1.3 - Vector to raster (v.to.rast) replaced v.dissolve. Vers. 1.4 - GRASS GIS was set aside; data was exported to file (ogr2ogr) and converted to raster using GDAL Python Bindings.
MOD 5	Select data related with Module 5 and import it to GRASS GIS (v.in.ogr) > Buffer calculation using default thresholds (v.buffer) > dissolve polygons (v.dissolve)	Vers. 1.2 - Same as Version 1.1. Vers. 1.3 - Vector to raster (v.to.rast) replaced v.dissolve. Vers. 1.4 - GRASS GIS was set aside; ST_Buffer replaced v.buffer; data was exported to file (ogr2ogr) and converted to raster using GDAL Python Bindings.
MOD 6	Select Polygon OSM Features related with Module 6 that intersects with Point OSM Features and import it to GRASS GIS (v.in.ogr) > dissolve polygons (v.dissolve); Select Polygon OSM Features related with Module 6 that not intersects with Point OSM Features and import it to GRASS GIS (v.in.ogr) > assign polygons to the LULC Class including residential and generalize polygons (v.dissolve).	Vers. 1.2 - Same as Version 1.1. Vers. 1.3 - Vector to raster (v.to.rast) replaced v.dissolve. Vers. 1.4 - GRASS GIS was set aside; ST_Buffer replaced v.buffer; data was exported to file (ogr2ogr) and converted to raster using GDAL Python Bindings.
PRIORITY Rule	The result of each module is a Vector Layer with several LULC Classes, so first thing to do is create a layer for each LULC Class (Geopandas) > these layers were compared to each other in an iterative process, which performs the intersection between one layer and the others with lower hierarchical rank, removing the common areas from the second one (v.overlay was used)	Vers. 1.2 - Same as Version 1.1. Vers. 1.3 - The result of each module is a list with raster files representing the presence of a certain class; r.patch was used to obtain a single raster for each LULC class; r.patch was used to apply the priority rule and generate the final result. Vers. 1.4 - The result of each module is a list with raster files representing the presence of a certain class; these files were converted to Numpy Arrays; arrays associated with the same LULC class were summed up to obtain an Array for each LULC class; these Arrays were compared to each other in an iterative process: At each iteration, considering each of the Arrays (ARRAY Y) and the Arrays with a lower hierarchical order (ARRAYS E), the values of each ARRAY E are replaced by NULL when, in the same position, the ARRAY Y value is greater or equal to one.

[a]For more information on how the different OSM2LULC Versions are implemented, its documentation can be found at https://github.com/jasp382/gasp/tree/master/gasp/osm2lulc

Execution time results

Figure 6 shows in graph plots: a) the execution time of all implementations; b) graph zoom-ins to provide details on the 12 min-time slot, so that a more detailed analysis may be done regarding the faster versions; and c) a closer look at the performance of Versions 1.3 and 1.4 considering spatial resolutions of 10 m, 5 m, and 2 m.

Results obtained showed that Version 1.1 is less efficient in all study areas compared to other versions. One can also see that the processing time is particularly high for Milan dataset (approximately 3 h00), where data size is almost the double of data available for London (see Table 3). This showed that this version was particularly sensitive to the input data size. Processing time got significantly reduced from Version 1.1 to Version 1.2, and even more from the latter to Versions 1.3 and 1.4, in which raster data is used in the priority rule application. In Version 1.2, processing time for Milan was reduced to approximately 7 min (i.e. an improvement of 96%), while for London it was reduced from a little more than 11 min to only approximately 2 min (i.e. an improvement of 80%). Similar proportional improvements were observed for Lisbon and Coimbra. Version 1.3 (Fig. 6c) enabled processing time reduction for Milan to less than 3 min with the three considered spatial resolutions (10 m, 5 m, and 2 m). Moreover, differences in processing time between these were very small. Processing time for the other study areas was between 30 s and 43 s

Table 6 Comparing performance of GRASS GIS in MODULE 2 with other alternatives

		Coimbra	Lisbon	London	Milan
Polygon generalization (dissolve) in MODULE 2	Reference value				
	GRASS GIS 7.6	00:03:11	00:07:35	00:08:55	02:53:20
	v.dissolve replaced by similar tools in other GIS Software				
	ArcGIS 10.6	00:00:38	00:00:34	00:00:52	00:03:31
	PostGIS 2.5	00:00:31	00:00:47	00:00:52	00:05:04
	Spatialite	00:59:00	> 1 h	> 1 h	> 1 h
	SAGA GIS 7.2	00:35:39	> 1 h	> 1 h	> 1 h
Buffering with generalization of buffer polygons in MODULE 2	Reference value				
	GRASS GIS 7.6	00:03:24	00:07:55	00:09:14	02:55:05
	v.buffer and v.dissolve replaced by buffer tool with dissolve option in other GIS Software				
	ArcGIS 10.6	00:00:27	00:00:30	00:00:33	00:00:59
	PostGIS 2.5	00:00:10	00:00:13	00:00:13	00:01:12
	Spatialite	00:00:28	00:00:48	00:01:06	00:11:38
	SAGA GIS 7.2	00:25:34	00:43:56	> 1 h	> 1 h

- except for Coimbra, where it took a little more than 1 min for the 2 m spatial resolution version. Version 1.4 showed to be very fast when compared to all the other implementations, taking always less than 40 s to obtain results, and no significant time differences occurred when using different spatial resolutions.

The analysis of time consumed in each part of the process (Fig. 7) allowed to conclude that in Version 1.1, MODULE 2 is the one consuming most of the time. If the performance of each tool used in MODULE 2 is analyzed in more detail, it can be seen that the GRASS GIS v.dissolve tool has an impact in the performance of the whole process, and in particular of MODULE 2, as it proved to be particularly slow when processing a greater number of features (6473 OSM features in Coimbra; 9461 in Lisbon; 12,005 in London; 38,076 in Milan).

In fact, polygon generalization operations are time consuming if performed with FOSS4G, which is a limitation when compared to some proprietary software packages, such as ArcGIS. Table 6 shows time taken in MODULE 2, if v.dissolve tool was replaced by analogue tools from other software, including ArcGIS. This shows that time taken by ArcGIS was much smaller when compared to all others - except PostGIS, which presented a better value than ArcGIS for Coimbra, the same value for London, and closer values for Lisbon and Milan when comparing differences between ArcGIS and the others.

In MODULE 2, v.dissolve tool is used to generalize polygons obtained with v.buffer tool, as this tool does not permit to keep the alphanumeric attributes associated with the lines used to generate the buffers. Tools to generate buffers in other GIS software provide this possibility. In the case of ArcGIS and PostGIS, running the buffer tool with the option “dissolve polygons” is more efficient than running these tools sequentially (Table 6).

In Version 1.2, v.buffer and v.dissolve were eliminated from the MODULE 2 workflow and replaced by PostGIS tools, since it proved to be the most efficient in performing tasks of distance calculation and buffering with polygon generalization (Table 6 and Table 7). The combined use of PostGIS tools ST_Distance and ST_DWithin allowed a rather efficient calculation of the distance between a set of input features and the closest entity of another layer (equivalent to the v.distance of GRASS GIS) (i.e. about 1 s in all study areas). In addition, the combined use of ST_Buffer, ST_Collect, and ST_UnaryUnion (equivalent to GRASS GIS v.dissolve) was also quite efficient in creating and generalizing a buffer built around a set of features. The combined use of ST_UnaryUnion and ST_Collect explains most of the efficiency gains, since this combined use leaves on our side the choice of how many geometries should be dissolved at once, hence allowing us to better control the memory size and CPU time. The integration of these PostGIS functions into MODULE 2 of Version 1.2 justifies the amplitudes verified between the two OSM2LULC implementations considered (Table 7).

However, the ability of Versions 1.3 and 1.4 to be even faster is related to the use of the raster data model. In these implementations, at a given moment, vector data was converted to raster, making the outputs of each module to be a set of raster files. Consequently, the hierarchical approach that seeks to solve subsistent inconsistencies and aggregates each class in a final layer was

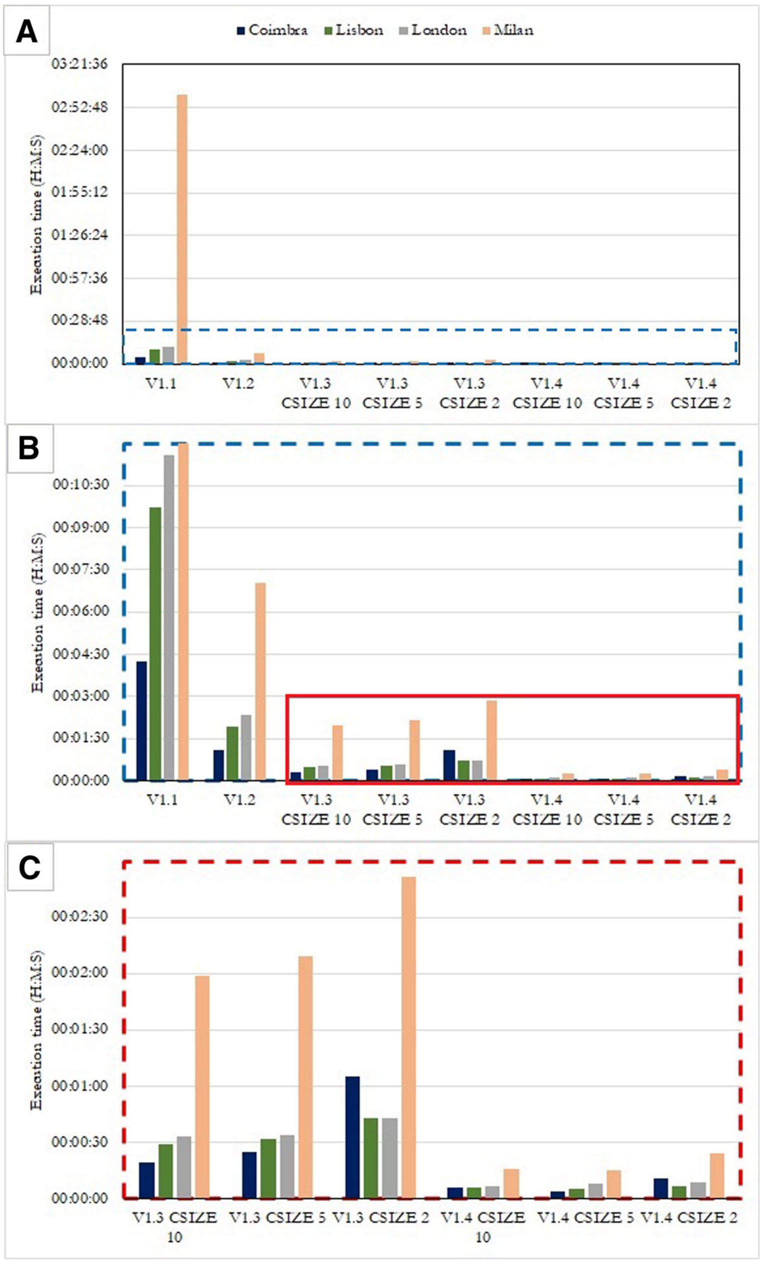
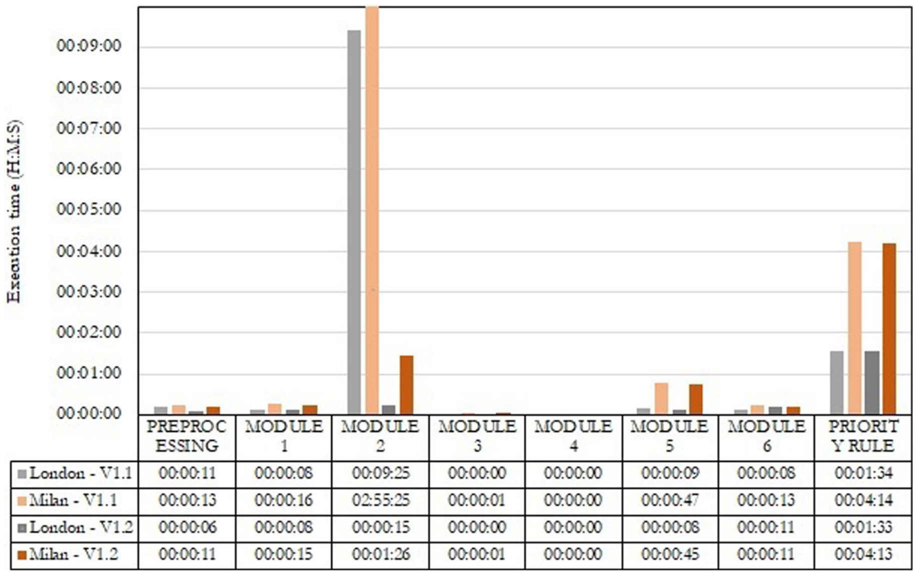


Fig. 6 Overview of the execution time of: **a**) all different versions; **b**) zoom-ins to provide details on the 12-min time slot; and **c**) an overview of results for Versions 1.3 and 1.4 for spatial resolutions of 10 m, 5 m, and 2 m

performed using raster files, which performed much faster (Fig. 8). In the specific case of Version 1.3, the performance enhancement also depends on the fact that this specific process is done by using r.patch tool, unlike Version 1.1 and 1.2, in which the implementation of the hierarchical approach is performed through several executions of v.overlay tool within a loop.

The way the hierarchical rule is implemented also explains why Version 1.4 was faster than 1.3 (Fig. 9). At this stage of processing, Version 1.3 works with r.patch tool and Version 1.4, on the contrary, operates with a tool we have developed (r.numpy.patch), which uses GDAL to convert raster files into Numpy Arrays and Numpy data structures to apply the priority rule. In Version 1.3, r.patch is used several times (to merge all rasters related with one particular LULC class into a single one; to apply the priority rule and producing the final result), which generates several temporary files throughout the process. Instead, in Version 1.4, with r.numpy.patch, no temporary files are produced; all data and all priority



	PREPROCESSING	MODULE 1	MODULE 2	MODULE 3	MODULE 4	MODULE 5	MODULE 6	PRIORITY RULE
■ London - V1.1	00:00:11	00:00:08	00:09:25	00:00:00	00:00:00	00:00:09	00:00:08	00:01:34
■ Milan - V1.1	00:00:13	00:00:16	02:55:25	00:00:01	00:00:00	00:00:47	00:00:13	00:04:14
■ London - V1.2	00:00:06	00:00:08	00:00:15	00:00:00	00:00:00	00:00:08	00:00:11	00:01:33
■ Milan - V1.2	00:00:11	00:00:15	00:01:26	00:00:01	00:00:00	00:00:45	00:00:11	00:04:13

Fig. 7 Execution time in each part of the procedure for London and Milan datasets using Versions 1.1 and 1.2

rule processes are respectively stored and done using only RAM memory, being the final result the only persisted file.

Despite the strengths of r.numpy.patch tool, it is important to point out that this tool still needs further development to properly manage memory usage. Numpy structures are stored in memory, so there is the possibility of breaking the execution of the program if the entire memory is taken. This indeed happens when the defined inputs have a geographical extent greater than those presented in this study, particularly if the cell size is very small. Therefore, the consolidation of Version 1.4 of OSM2LULC implies the development of memory control and management mechanisms to avoid error messages when the user considers a large extent area. This management system should restrict the size of the area of interest that can be defined by the user, and/or segment the area of interest running the process individually for each part.

Nonetheless, there are other factors that may help explaining why Version 1.4 is faster: (i) in Version 1.3, data must be imported into GRASS GIS and then converted to raster, which is a time-consuming task; (ii) in MODULE 2 of Version 1.3, buffer calculation includes the generalization of existing polygons, while in module 2 of Version 1.4, buffer calculation is performed without generalization. This causes PostGIS to take a longer time to return the results in Version 1.3. However, if the generalization operation is skipped, GRASS GIS takes even longer to import the output.

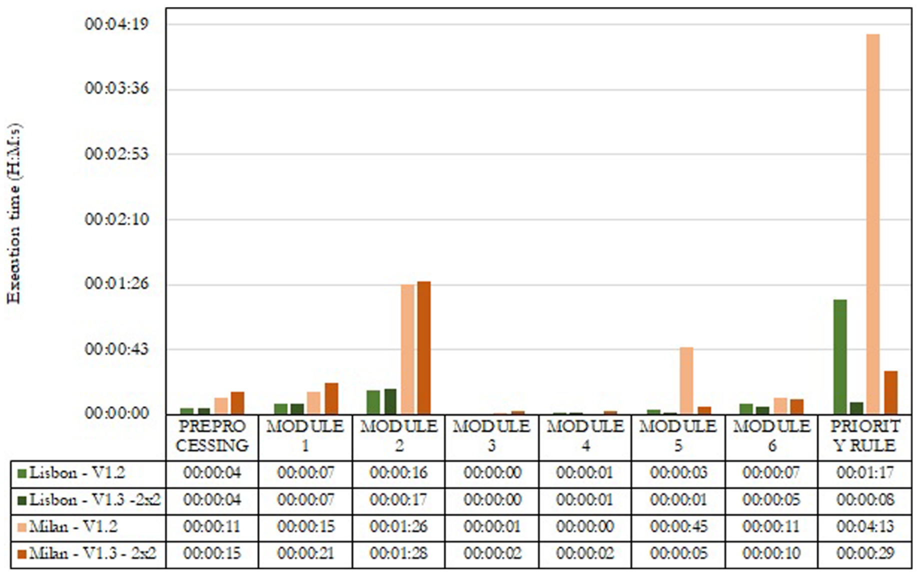
Only MODULE 2 and the PRIORITY RULE were debated in this discussion, since they concern parts of the process that consume more time. In fact, the remaining modules do not significantly affect the performance of the various implementations, namely MODULES 3, 4 and 5, which is explained by the reduced number of OSM features processed by them (Fig. 10).

Positional distortions in the results

All versions described in the article implement the same algorithm. However, on one hand, the use of different tools available in different software packages generate output differences, and, on the other hand, in some versions and modules raster data is used (with different pixel sizes) instead of vector, which also produced

Table 7 Gains of using PostGIS instead of GRASS GIS in MODULE 2 for Milan study area

MODULE 2	GRASS GIS	POSTGIS
Import Roads and Buildings into GRASS GIS	00:00:15	–
Near analysis - distance between roads and buildings	00:00:05	00:00:01
Buffer analysis	02:55:05	00:01:12
Import result of buffer analysis into GRASS GIS	–	00:00:13
Total	02:55:25	00:01:26

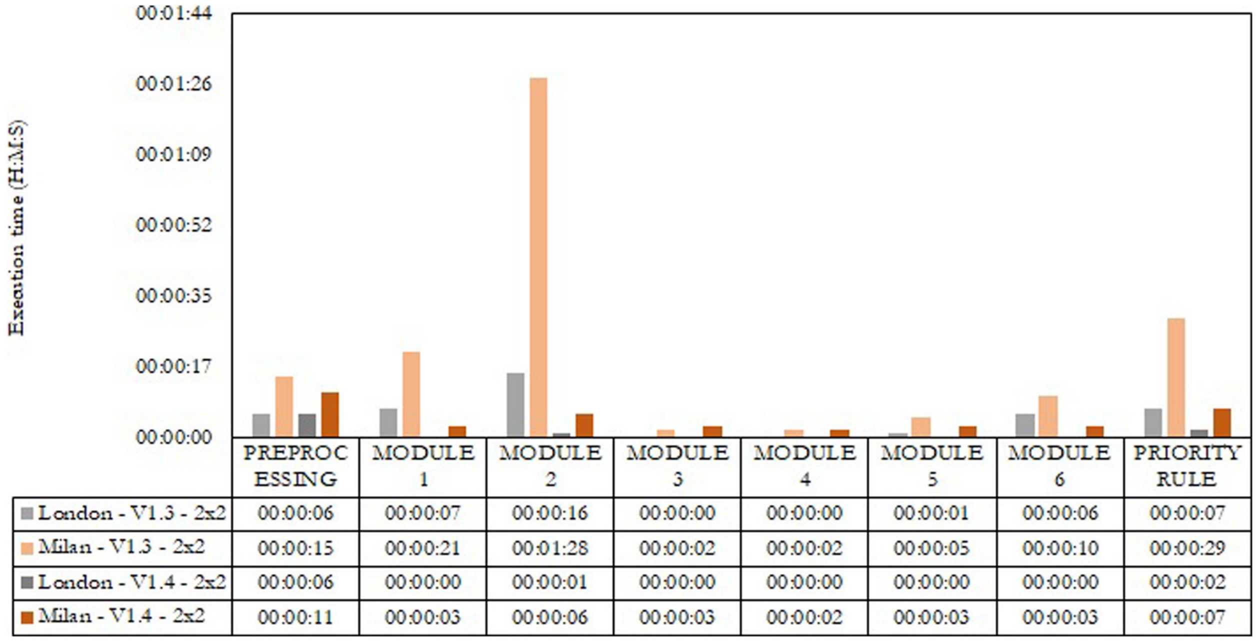


	PREPRO CESSING	MODULE 1	MODULE 2	MODULE 3	MODULE 4	MODULE 5	MODULE 6	PRIORIT Y RULE
🟩 Lisbon - V1.2	00:00:04	00:00:07	00:00:16	00:00:00	00:00:01	00:00:03	00:00:07	00:01:17
🟩 Lisbon - V1.3 -2x2	00:00:04	00:00:07	00:00:17	00:00:00	00:00:01	00:00:01	00:00:05	00:00:08
🟧 Milan - V1.2	00:00:11	00:00:15	00:01:26	00:00:01	00:00:00	00:00:45	00:00:11	00:04:13
🟧 Milan - V1.3 - 2x2	00:00:15	00:00:21	00:01:28	00:00:02	00:00:02	00:00:05	00:00:10	00:00:29

Fig. 8 Execution time consumption in each part of the procedure - Version 1.2 and 1.3; Lisbon and Milan datasets

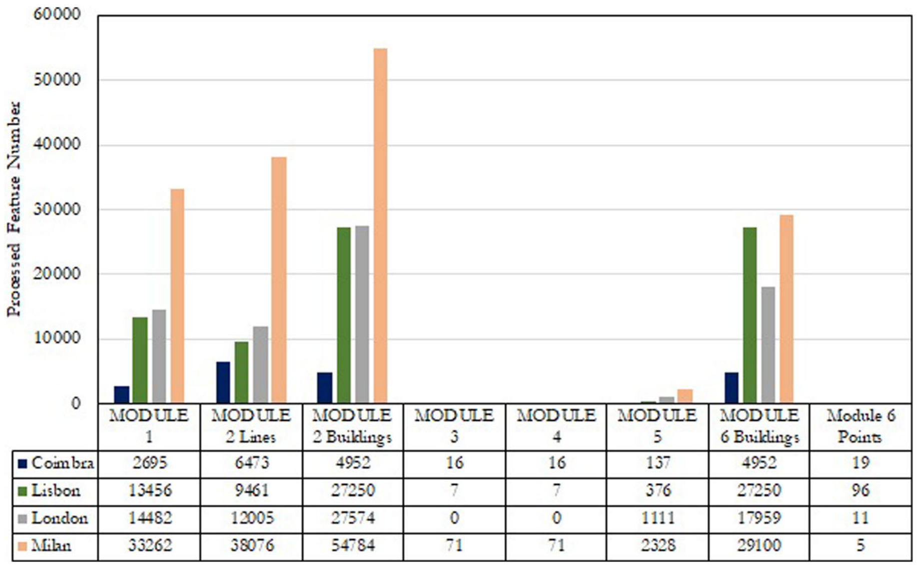
differences in the final outputs. To assess the magnitude of such differences, positional distortions in the results obtained with the different versions of OSM2LULC were analysed for each study area considered. Table 8 shows the percentage of areas not coincident with the results obtained with Version 1.1. It can be seen that results obtained with Version 1.2 are almost identical to the ones obtained with Version 1.1, with differences smaller than 0.1% for the four study areas. These differences are due to the use of different tools in the versions. For example, in Version 1.1 MODULE 2 uses only GRASS GIS tools, while in Version 1.2 PostGIS tools are used. This results in minor differences in the buffer shapes, mainly at their ends.

As expected, differences between Version 1.1 and Versions 1.3 and 1.4 are larger, as raster results are obtained with these versions. Such differences depend mainly on the spatial resolution used for raster files, decreasing with an increase in spatial resolution. For all study areas, differences between 8% and 10% were obtained when



	PREPROC ESSING	MODULE 1	MODULE 2	MODULE 3	MODULE 4	MODULE 5	MODULE 6	PRIORITY RULE
⬜ London - V1.3 - 2x2	00:00:06	00:00:07	00:00:16	00:00:00	00:00:00	00:00:01	00:00:06	00:00:07
🟧 Milan - V1.3 - 2x2	00:00:15	00:00:21	00:01:28	00:00:02	00:00:02	00:00:05	00:00:10	00:00:29
⬛ London - V1.4 - 2x2	00:00:06	00:00:00	00:00:01	00:00:00	00:00:00	00:00:00	00:00:00	00:00:02
🟧 Milan - V1.4 - 2x2	00:00:11	00:00:03	00:00:06	00:00:03	00:00:02	00:00:03	00:00:03	00:00:07

Fig. 9 Execution time consumption in each part of the procedure – Version 1.3 and 1.4; London and Milan datasets



	MODULE 1	MODULE 2 Lines	MODULE 2 Buildings	MODULE 3	MODULE 4	MODULE 5	MODULE 6 Buildings	Module 6 Points
■ Coimbra	2695	6473	4952	16	16	137	4952	19
■ Lisbon	13456	9461	27250	7	7	376	27250	96
■ London	14482	12005	27574	0	0	1111	17959	11
■ Milan	33262	38076	54784	71	71	2328	29100	5

Fig. 10 Number of Features processed in each MODULE by study area

considering a cell size of 10 m, decreasing to values between 4% and 6% when considering a cell size of 5 m, and to around 2% when considering a cell size of 2 m. Using London as an example, Fig. 11 illustrates differences caused by cell size definition, found in all study areas.

Table 9 shows the contribution of each class to the areas of disagreement between version 1.1 and versions 1.3/1.4 for the 2 m spatial resolution outputs. It can be seen that class 1.2 and the regions with no data in OSM are, by far, the ones contributing more to the disagreements. Class 1.2 includes all roads, which are narrow and long linear features that, when converted to the raster structure, in order to adapt to the pixel shape end up increasing their width a little bit (depending of course on the size of the pixel). As this occurs all over along the roads, it ends up increasing considerably the total area of features. Regarding regions with no data in OSM, when vector features are converted to the raster format their limits will occupy neighbouring regions, which frequently are not occupied by any other feature in OSM. This is particularly evident for roads, and it was observed that there is positive correlation between the contribution of roads within class 1.2 to the disagreement and the contribution of regions with no data.

The identified differences between vector and raster outputs with the three spatial resolutions, correspond to normal differences between data represented with vector and raster data structure. Our concern here is, however, if the use of versions providing results at higher speed may compromise their use for some applications. Even though, differences in percentages are not as considerable (see Table 8); in fact, they may still correspond to a few hundreds of hectares (for Milan study area, with a 2 m spatial resolution, differences reached more than 600 ha). Therefore, for each application, it is necessary to check if raster LULC maps may be used or if versions providing vector results should be used instead.

Table 8 Disagreement (in percentage relative to the data in OSM and area in ha) between Version 1.1 and Versions 1.2, 1.3 and 1.4, these last with 10 m, 5 m and 2 m spatial resolution

Disagreement area	Version 1.2	Version 1.3/1.4 10 m	Version 1.3/1.4 5 m	Version 1.3/1.4 2 m
Version 1.1 Coimbra	0.01% 0.86 ha	10.05% 902.83 ha	5.71% 503.96 ha	2.20% 191.24 ha
Version 1.1 Lisbon	0.04% 3.38 ha	8.31% 716.80 ha	4.86% 415.19 ha	2.01% 170.70 ha
Version 1.1 London	0.05% 4.18 ha	11.17% 963.93 ha	6.49% 553.26 ha	2.58% 217.41 ha
Version 1.1. Milan	0.05% 15.78 ha	8.76% 2834.48 ha	5.00% 1601.09 ha	1.94% 614.55 ha

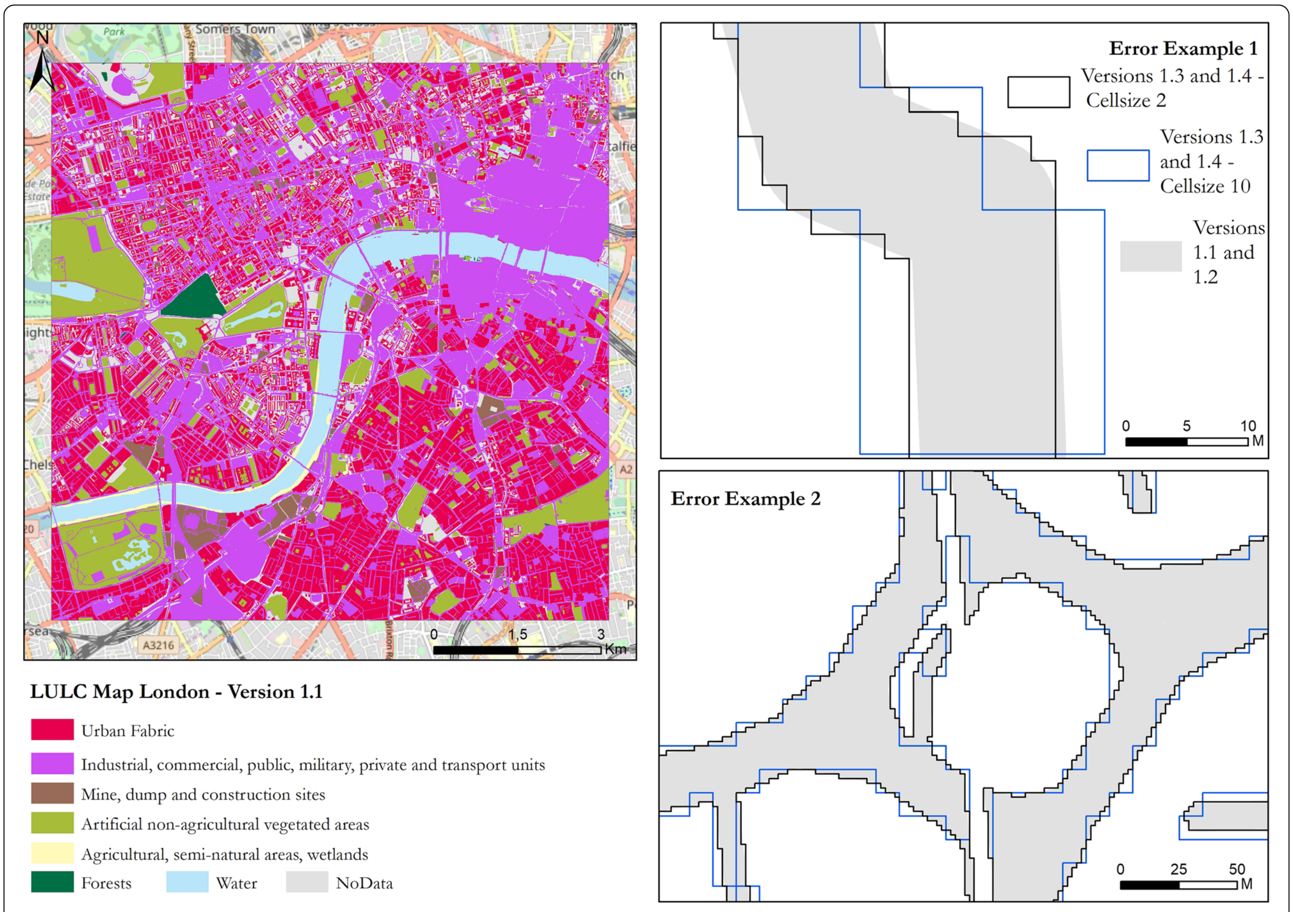

LULC Map London - Version 1.1
Urban Fabric
Industrial, commercial, public, military, private and transport units
Mine, dump and construction sites
Artificial non-agricultural vegetated areas
Agricultural, semi-natural areas, wetlands
Forests
Water
NoData
0 1,5 3 Km
Error Example 1
Versions 1.3 and 1.4 - Cellsize 2
Versions 1.3 and 1.4 - Cellsize 10
Versions 1.1 and 1.2
0 5 10 M
Error Example 2
0 25 50 M

Fig. 11 Left: results obtained for the London study area with Version 1.1. Right: examples of results obtained with Versions 1.3 and 1.4 with 2 and 10 spatial resolution overlapped with the results obtained with version 1.1

Conclusions and further work

OSM2LULC software package was developed to convert OSM data into LULC maps. In order to maximize performance, four versions of the underlying algorithm were implemented using different FOSS4G software tools. The performance of the four versions was tested by considering four study areas with different data volumes and area sizes. Results showed that Version 1.4 was the one showing the overall best performance - indeed, when applied to the four study areas, it always took less than 1 min; this was a substantial improvement when compared with the execution time taken by Version 1.1 - for example, it took more than three hours when dealing with Milan study area. Version 1.3 also showed to have a relatively good performance; but, when data volume increases, time required for the conversion

Table 9 Contribution of each LULC class (in %) to the disagreements between Versions 1.1 and 1.3/1.4 (2 m cell size) for all study areas

LULC Class	Coimbra	Lisbon	London	Milan
1.1 - Urban Fabric	6.11	23.14	14.11	14.10
1.2 - Industrial, comercial, public, militar, private and transport units	43.89	43.69	43.77	42.87
1.3 - Mine, dump and construction sites	0	0.14	0.69	0.56
1.4 - Artifical non-agricultural vegetated areas	0.61	2.66	2.23	3.14
2 - Agricultural, semi-natural areas	1.47	1.77	0.34	2.7
3 - Forests	0.43	0.07	0.06	0.08
5 - Water	2.93	0.61	0.86	0.64
NoData	44.56	27.92	0.64	35.75

process may increase too (for Milan study area, with 196 MB, it took already more than 2 min). This difference occurred because in Version 1.3 GRASS GIS was used and this requires the use of data in GRASS vector format, while in Version 1.4 GRASS GIS was not used at all. Unlike ESRI Shapefile data structure, the GRASS Vector data model includes topology describing spatial relations between feature location and geometry [23]. Therefore, besides the need to convert data into GRASS Vector format, GRASS GIS explores the topological relations between data, which may be very useful in some cases taking additional time though, affecting therefore the overall performance. In this test case, GRASS GIS confirmed to provide results with high quality [24], however the efficiency of some tools may still to be improved.

Currently, the efficiency of Version 1.4 is only constrained by the geospatial extent of the study area and the considered cell size, and is not dependent on the amount of data available in OSM. Despite the differences obtained by using raster data and by changing cell sizes, results with smaller cell sizes (such as 2 m) can be used for many applications. The main limitation of this version is related to the dimension of the arrays used when applying the hierarchical approach. In order to minimize this limitation and to further optimize the performance of Version 1.4, the execution of each module will be performed in parallel leveraging the multiprocessing Python package, since the various modules are completely independent, with no data sharing between them. In addition, when applying the hierarchical approach to very large extent areas, r.numpy.patch will subdivide the whole area into smaller regions to take advantage of the parallel processing. By default, Python is limited to use a single CPU-core due to the Python Global Interpreter Lock (GIL), even when a multi-threading approach is used. However, it is possible to split the workflow of a Python program into multiple processes instead of threads. With multiprocessing, each created process runs separately using a single CPU-core and has its own Python interpreter and RAM memory space. This will enable OSM2LULC to be applied to much larger areas and to take full advantage of CPU's processing capabilities.

Even though results provided by Versions 1.3 and 1.4 are not very different from the ones obtained with the other versions (i.e. the ones with final results in vector format), for applications where positional accuracy is critical vector results should be used instead. Therefore, efforts will be made in the future to improve the efficiency of OSM2LULC versions that provide vector outputs, namely Version 1.2. Tests made so far with FOSS4G suggests the possibility to develop more efficient methodologies integrating several FOSS4G tools, along with input data segmentation and parallel processing, using a multiprocessing approach. The modules could be executed in parallel using only PostGIS tools; the priority rule could be applied using GRASS GIS and a strategy based on geographic data segmentation and parallelization with the multiprocessing Python module. PostGIS will progressively replace GRASS GIS because it is faster in buffering and polygon aggregation tasks, but the latter will remain the primary solution when applying the priority rule. Our experience tells us that GRASS GIS is significantly faster when working with less data, even if there are other tasks running in parallel. Thus, dividing data into different parts and running v.overlay for each part in parallel seems to be the most promising solution.

Should be noted that the parallel processing of data can only be applied to the different modules, and not within each module. Otherwise we could end up losing geospatial relations and geometric attributes of the data.

For the time being, it was possible to implement OSM2LULC using only FOSS4G tools. However, in order to obtain an overall better performance, it is necessary to identify which tools have the best performance in each step of the algorithm, and integrate them considering an interoperability approach. This will allow to overcome identified limitations and eventual bugs of tools available in the software packages. Furthermore, a substantial comparative study of tools available, which may work out time consuming and requires prior experience with such software packages, is also needed.

As stated in Section 1, the underlying algorithm of OSM2LULC has revealed some limitations and improvements are needed. Thus, it is envisaged the implementation of further logical rules based on expected thematic-geospatial relationships between the OSM features involved in the process, as well as the use of other data sources that may contribute towards a more accurate thematic and positional validation of OSM data.

Abbreviations

CPU: Central Processing Unit; EPSG: European Petroleum Survey Group; FOSS4G: Free and Open Source Software for Geospatial; FSF: Free Software Foundation; GASP: GeoData Algorithms for Spatial Problems; GDAL: Geospatial Data Abstraction Library; GIL: Global Interpreter Lock; GIS: Geographic Information System; GRASS GIS: Geographic Resources Analysis Support System; LTS: Long Term Support; LULC: Land Use Land Cover; OSM: OpenStreetMap; OSM2LULC: OpenStreetMap to Land Use/Land Cover; RAM: Random Access Memory; SAGA GIS: System for Automated Geoscientific Analyses; SRS: Spatial Reference System; VGI: Volunteered Geographic Information

Acknowledgments

The study has been partly supported by the Portuguese Foundation for Science and Technology (FCT) under project grant UID/MULTI/ 00308/2019 (J.P., C.C.F., J.A.).

Authors' contributions

CCF and JP contributed to the development of the OSM2LULC algorithm, J. P implemented the code of all versions and run the algorithm for several case studies, JP was responsible for the comparison of results, JE contributed to the implementation choices, all authors contributed to discussions on the article preparation and writing. All authors read and approved the final manuscript.

Funding
Not applicable.

Availability of data and materials
Datasets generated and/or analysed during the current study are not publicly available but are available from the corresponding author on reasonable request.

Competing interests
The authors declare that they have no competing interests.

Author details
[1]Institute for Systems Engineering & Computers at Coimbra (INESCC), Rua Sílvio Lima -Edifício DEEC, S.3.4, 3030-290 Coimbra, Portugal. [2]Department of Mathematics – Geomatic Engineering Group, Faculty of Science & Technology, University of Coimbra, Apartado 3008, 3001-501 Coimbra, Portugal. [3]Setúbal School of Technology, Campus do IPS, Estefanilha, 2914-504 Setúbal, Portugal. [4]Department of Informatics Engineering, Faculty of Science & Technology, CISUC – University of Coimbra, Rua Sílvio Lima, 3030-290 Coimbra, Portugal. [5]Institute for Systems Engineering & Computers, Research and Development at Lisboa (INESC-ID), Rua Alves Redol, 9 (office 435), 1000-029 Lisbon, Portugal.

Received: 22 March 2019 Accepted: 31 July 2019
Published online: 01 October 2019

References

1. Antoniou V, Fonte C, See L, Estima J, Arsanjani J, Lupia F, Minghini M, Foody G, Fritz S. Investigating the feasibility of geo-tagged photographs as sources of land cover input data. ISPRS Int J Geo Inf. 2016;5:64.
2. Arpaci A, Malowerschnig B, Sass O, Vacik H. Using multi variate data mining techniques for estimating fire susceptibility of Tyrolean forests. Appl Geogr. 2014;53:258–70.
3. Arsanjani J, Helbich M, Bakillah M, Hagenauer J, Zipf A. Toward mapping land-use patterns from volunteered geographic information. Int J Geogr Inform Sci. 2013;27:2264–78.
4. Chen W, Peng J, Hong H, Shahabi H, Pradhan B, Liu J, Zhu A, Pei X, Duan Z. Landslide susceptibility modelling using GIS-based machine learning techniques for Chongren County, Jiangxi Province, China. Sci Total Environ. 2018;626:1121–35.
5. Estima J, Fonte CC, Painho M. Comparative study of land use/cover classification using Flickr photos, satellite imagery and Corine land cover database. In: Proceedings of the 17th AGILE conference on geographic information science. Castellón: Association of Geographic Information Laboratories in Europe (AGILE); 2014. p. 3–6.
6. Feddema JJ, Oleson KW, Bonan GB, Mearns LO, Buja LE, Meehl GA, Washington WM. The importance of land-cover change in simulating future climates. Science. 2005;310:1674–8.
7. Fonte CC, Martinho N. Assessing the applicability of OpenStreetMap data to assist the validation of land use/land cover maps. Int J Geogr Inf Sci. 2017; 31:12: 1–19.
8. Fonte CC, Minghini M, Antoniou V, See L, Patriarca J, Brovelli M, Milcinski G. Automated methodology for converting OSM data into a land use/land cover map. In: 6th international conference on cartography & GIS. Albena: Bulgarian Cartographic Association; 2016. p. 13–7.
9. Fonte CC, Minghini M, Patriarca J, Antoniou V, See L, Skopeliti A. Generating up-to-data and detailed land use and land cover maps using OpenStreetMap and GlobeLand30. ISPRS Int J Geo Inf. 2017;6:125. https://doi.org/10.3390/ijgi6040125.
10. Fonte CC, Patriarca J, Minghini M, Antoniou V, See L, Brovelli MA. Using OpenStreetMap to create land use and land cover maps: development of an application. In: Campelo C, Bertolotto M, Corcoran P, editors. Volunteered geographic information and the future of geospatial data. Hershey. ISBN: 9781522524465: IGI Global; 2017. https://doi.org/10.4018/978-1-5225-2446-5.ch007.
11. Fritz S, McCallum I, Schill C, Perger C, See L, Schepaschenko D, Velde M, Kraxner F, Obersteiner M. Geo-wiki: an online platform for improving global land cover. Environ Model Softw. 2012;31:110–23.
12. Fuggetta J. Open source software: na evaluation. J Syst Software. 2003;66: 77–90.
13. Ganesh A. Validating OpenStreetMap. 2017 https://2017.stateofthemap.org/2017/validating-openstreetmap/ (accessed March 10, 2019).
14. Gauci A, Abela J, Austad M, Cassar L, Adami K. A machine learning approach for automatic land cover mapping from DSLR images over the Maltese islands. Environ Model Softw. 2018;99:1–10.
15. Gay J. Free software, free society: selected essays of Richard Stallman. Boston: GNU Press; 2002.
16. Hollmann R, Merchant CJ, Saunders R, Downy C, Buchwitz M, Cazenave A, Chuvieco E, Defourny P, de Leeuw G, Forsberg R, Holzer-Popp T, Paul F, Sandven S, Sathyendranath S, van Roozendael M, Wagner W. The ESA climate change initiative: satellite data Records for Essential Climate Variables. Bull Am Meteorol Soc. 2013;94:1541–52.
17. Lai C, Shao Q, Chen X, Wang Z, Zhou X, Yang B, Zhang L. Flood risk zoning using a rule mining based on ant colony algorithm. J Hydrol. 2016;542:268–80.
18. Laurent A. Understanding open source & free software licensing. Sebastopol: O'Reilly; 2004. First Edition
19. Lindberg V. Intellectual property and open source – a practical guide to protecting code. Sebastopol: O'Reilly Media; 2008. First Edition
20. Mapbox. Validating OpenStreetMap – Mapping Guides. 2018. https://labs.mapbox.com/mapping/validating-osm/ (accessed March 10, 2019).
21. Martinelli L. Can we validate every change on OSM? 2018 https://2018.stateofthemap.org/2018/T079-Can_we_validate_every_change_on_OSM_/ (accessed March 10, 2019)
22. Mooney P, Minghini M. A Review of OpenStreetMap Data. In: Foody G, See L, Fritz S, Mooney P, Olteanu-Raimond A-M, Fonte CC, Antoniou V, editors. Mapping and the Citizen Sensor. London: Ubiquity Press; 2017. p. 37–59.
23. Neteler M, Mitasova H. Open source GIS: a GRASS GIS approach. New York: Springer; 2008.
24. Neteler M, Bowman M, Landa M, Metz M. GRASS GIS: a multi-purpose open source GIS. Environ Model Softw. 2012;31:124–30.
25. OSMWiki (2018) OSM Tasking Manager/Validating data. https://wiki.openstreetmap.org/wiki/OSM_Tasking_Manager/Validating_data (accessed March 10, 2019).
26. Patriarca J (2016) O Software Livre e de Código Aberto na Administração Pública - Dos mitos às questões de natureza legal, ética e de optimização de recursos públicos. Master Dissertation, University of Coimbra. Available at https://estudogeral.sib.uc.pt/handle/10316/30768. accessed March 10, 2019).
27. Phillips D. Unveiled - how legislation by license controls software access. New York: Oxford University Press; 2009.
28. Santos JG. GIS-based hazard and risk maps of the Douro river basin (North-Eastern Portugal). Geomatics Nat Hazards Risk. 2015;6:90–114.
29. Schultz M, Voss J, Auer M, Carter S, Zipf A. Open land cover from OpenStreetMap and remote sensing. Int J Appl Earth Obs Geoinf. 2017;63: 206–13.
30. See L, Schepaschenko D, Lesiv M, McCallum I, Fritz S, Comber A, Perger C, Schill C, Zhao Y, Maus V, Siraj M, Albrecht F, Cripriani A, Vakolyuk M, Garcia A, Rabia A, Singha K, Marcarini A, Kattenborn T, Hazarika R, Schepaschenko M, Velde M, Kraxner F, Obersteiner M. Building a hybrid land cover map with crowdsourcing and geographically weighted regression. ISPRS J Photogramm Remote Sens. 2015;103:48–56.
31. Steinhausen M, Wagner P, Narasimhan B, Waske B. Combining Sentinel-1 and Sentinel-2 data for improved land use and land cover mapping of monsoon regions. Int J Appl Earth Obs Geoinf. 2018;73:595–604.
32. Steiniger S, Bocher E. Na overview on current free and open source desktop GIS developments. Int J Geogr Inform Sci. 2009;23:1345–70.
33. Steiniger S, Hunter A. The 2012 free and open source software GIS software map – a guide to facilitate research, development, and adoption. Comput Environ Urban Syst. 2013;39:136–50.
34. Turner BL, Lambin E, Reenberg A. The emergence of land change science for global environmental change and sustainability. Proc Natl Acad Sci. 2007;104(52):20666–71.
35. Zhang X, Liu L, Wang Y, Hu Y, Zhang B. A SPECLib-based operational classification approach: a preliminary test on China land cover mapping at 30 m. Int J Appl Earth Obs Geoinf. 2018;71:83–94.