Edge-matching in data harmonisation

Hugo Ledoux
h.ledoux@tudelft.nl

Ken Arroyo Ohori
g.a.k.arroyoohori@tudelft.nl

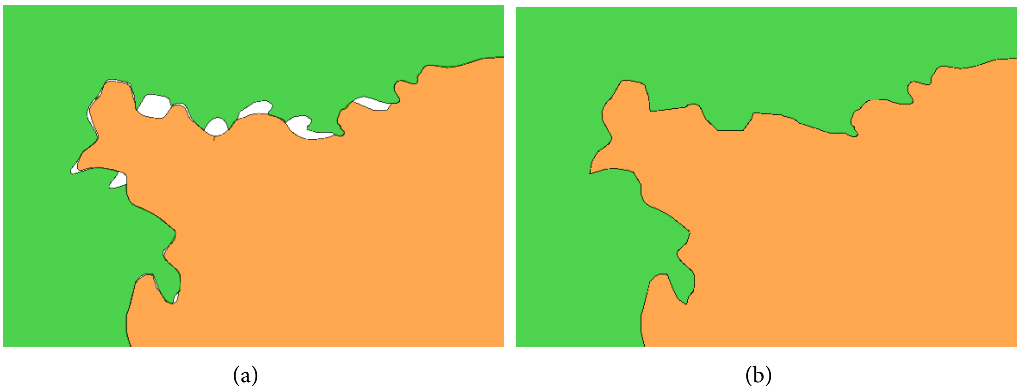
This technical report was written for the HUMBOLDT project (http://www.esdi-humboldt.eu/).

The code of the project is available at
https://github.com/tudelft-gist/pprepair

One of the main challenges of data harmonisation is that of edge-matching, ie managing the connections of geographical objects at international, regional or dataset-dependent boundaries, to ensure that objects on both sides are coherent. We present in this chapter the work that has been done during the HUMBOLDT project. First, a conceptually simple algorithm has been implemented and made freely available to the public as a Web Processing Service. During the testing of this service with various datasets, it appeared that the method used—snapping of vertices based on a user-defined threshold—was error-prone and often lead to geometries that were invalid. We have therefore developed a novel algorithm where vertices are not moved (no snapping is involved); instead gaps and overlaps between polygons are corrected by using a constrained triangulation as a supporting structure. We present in the paper our novel algorithm and our implementation, which is based on the stable and fast triangulator in CGAL. We also present some experiments we have made with real-world cross-boundary datasets in Europe, and we compare the two implementations. Our experiments demonstrate that our novel algorithm is highly efficient and permits us to avoid the tedious task of finding the optimal threshold for a dataset, for the polygons are properly edge-matched and we can prove that no gaps/overlaps are left.

1 Introduction

With the HUMBOLDT project, and within this book, the harmonisation of geographical information is mostly tackled from a data modelling point-of-view, but one should be aware that other



(a)

(b)

Figure 1: **(a)** Part of the polygons representing the Arribes del Duero Natural Park in Spain (orange) and the International Douro Natural Park in Portugal (green). Since the border is defined as a river, the two datasets do not match perfectly (there are gaps and overlaps). **(b)** The polygons after edge-matching has been successfully performed.

issues were also tackled during the project. This chapter discusses one of these: *edge-matching*, which is also part of a bigger process called *geometric conflation*. It refers to the management of the connections of geographical objects at boundaries (to ensure that objects on both sides are coherent), and it is one of the main challenge when dealing with datasets produced by different organisations or countries. It involves combining multiple datasets in order to make a new one, usually to improve either the spatial extent or the accuracy of the data (Lynch and Saalfeld, 1985). Yuan and Tao (1999) and Davis (n.a.) make a distinction between three types of conflation:

Horizontal conflation refers to edge-matching of neighbouring datasets to eliminate discrepancies at the border region. Country borders defined based on natural features of the terrain are a good example since their continuous nature basically ensures that independently produced data will not match at the border (Burrough, 1992). Figure 1 shows an area along the Spanish-Portuguese border with this problem.

Vertical conflation involves combining datasets covering the same area.

Internal conflation refers to the "cleaning" of a single dataset so that it does not contain gaps or that its polygons do not overlap.

As further explained in Section 3, the edge-matching problem has traditionally been tackled almost exclusively by using the concept of a *threshold* (a tolerance). In other words, if two objects (edges or vertices) are closer to each other than a given tolerance (which is usually defined by the user) then they are considered "equal" and can be *snapped* together so that they become the same object in the resulting dataset.

Two implementations of an edge-matching algorithm have been performed during the HUMBOLDT project. The first one, described in Section 2 and called the *Edge Matching Service* (EMS), is based on the threshold concept, is freely available as a Web Processing Service (WPS), and can be used for lines and polygons. During the evaluation of the first implementation of the EMS, we noticed that while snapping yields satisfactory results for simple problems, for complex ones it was often impossible or impractical to find a tolerance applicable to the whole dataset, and furthermore the output geometries of the EMS were often *invalid*. Such invalid geometries might not be visible to the user (for instance tiny gaps and overlaps might be remaining, or a line might self-intersect), but further processing with a GIS requires that datasets be valid. We review in Section 3 the previous edge-matching algorithms and we highlight the main pitfalls when snapping geometries.

To solve the problems caused by snapping and the use of a threshold, we have also developed a novel algorithm to perform edge-matching, and we have implemented it. It avoids the pitfalls of snapping but it is at this moment limited to polygons, ie the edge-matching of lines is not supported yet (although we plan to add it in the future). As explained in Section 4, our algorithm differs from the previous ones, since vertices of the geometries are never moved, ie no snapping of geometries and no thresholds are involved. Instead, we fill the gaps and fix the overlaps between datasets by using a *constrained triangulation* (CT) as a supporting structure and assigning values to triangles. This approach has in our opinion several advantages: (i) no user-defined tolerance needs to be defined (the triangles permit us to find matching polygons *locally*); (ii) we can control locally how the edges should be matched (in contrast to snapping, which often involves a global tolerance); (iii) we guarantee that the resulting edge-matched polygons will be valid. We report in Section 5 on our implementation of the algorithm (it is based on the stable and fast triangulator in CGAL[1]) and on the experiments we have made with some real-world datasets in Europe. Finally, we discuss in Section 6 the shortcomings of our method and future work.

2 Edge-matching as implemented in HUMBOLDT

The most common method for edge-matching is based on the concept that objects *approximately* match each other at their common boundaries (this approximation is based on a threshold). This implies that they should always be within a certain distance of each other along those borders. If, additionally, all parts further apart than this value are known not to be common boundaries, it is possible to snap together objects that are closer to each other than this threshold, while keeping the rest untouched. Most commercial GISs implement the method (eg ArcGIS, FME, GRASS and Radius Topology, albeit the algorithms used differ in the order in which points are snapped together and to what geometries: only to points or also to lines), and the INSPIRE Directive is explicit about the use of threshold (INSPIRE, 2008):

> *It will be to each "Thematic Working Group" to define the appropriate thresholds, if required, in a given data product specification, for each case of edge-matching.*

During the HUMBOLDT project, the EMS was developed and implemented. It permits us to edge-match both lines and polygons, and it is based on snapping of points (to other points, snapping between points and lines is not supported) that are within a user-defined threshold. It can be used for horizontal conflation of polygons (as shown in Figure 1), for vertical conflation of lines (as

[1] The Computational Geometry Algorithms Library: `http://www.cgal.org`

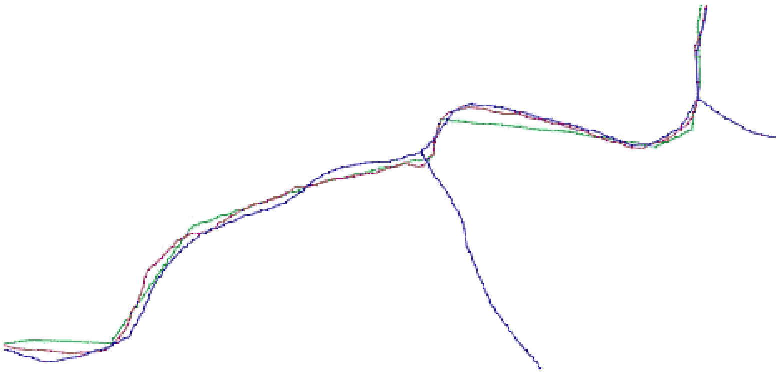


Figure 2: The administrative boundaries of Italy (scale 1:25,000), Ligure (scale 1:5,000) and of some river datasets (scale 1:10,000) do not fit and have to be edge-matched

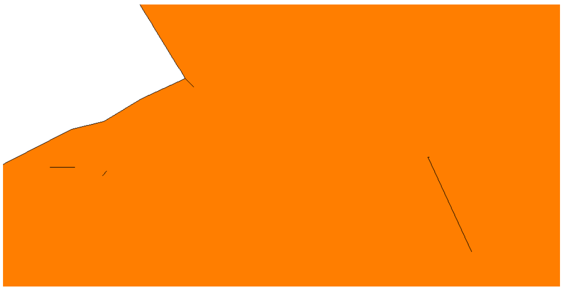


Figure 3: The Arribes del Duero Natural Park in Spain has several holes and spikes which causes its geometry to be valid.

shown in Figure 2), and for internal conflation. Indeed, several datasets for the HUMBOLDT scenarios contained "spiky" holes (as shown in Figure 3) and the EMS was used to remove them.

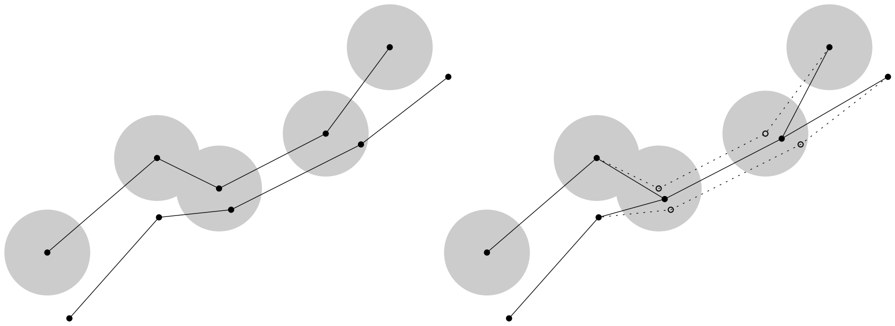
Moreover, when snapping geometries, the EMS allows us to either use one dataset as a *reference* dataset, or to distribute the errors equally between the datasets. The former implies that one of the two input datasets has higher accuracy, and therefore other datasets should be snapped to it; this can also be used when for instance the boundary between two countries is known and cannot be modified, other objects (provinces or regional boundaries) should be moved to fit the higher accuracy boundary, and not the other way around. The latter way of snapping is used when the accuracy of both datasets is the same, or is not known. Two or more points that are within a user-defined threshold will then be snapped "in the middle" as Figure 4 illustrates.

A WPS implementation of the EMS is available[2]. It permits users to use a desktop GIS as there are plug-ins for both OpenJUMP GIS[3] and uDig[4]; Figure 5 shows the EMS WPS plug-in for OpenJUMP in action

[2] Details can be obtained at `http://community.esdi-humboldt.eu/projects/show/ems`.

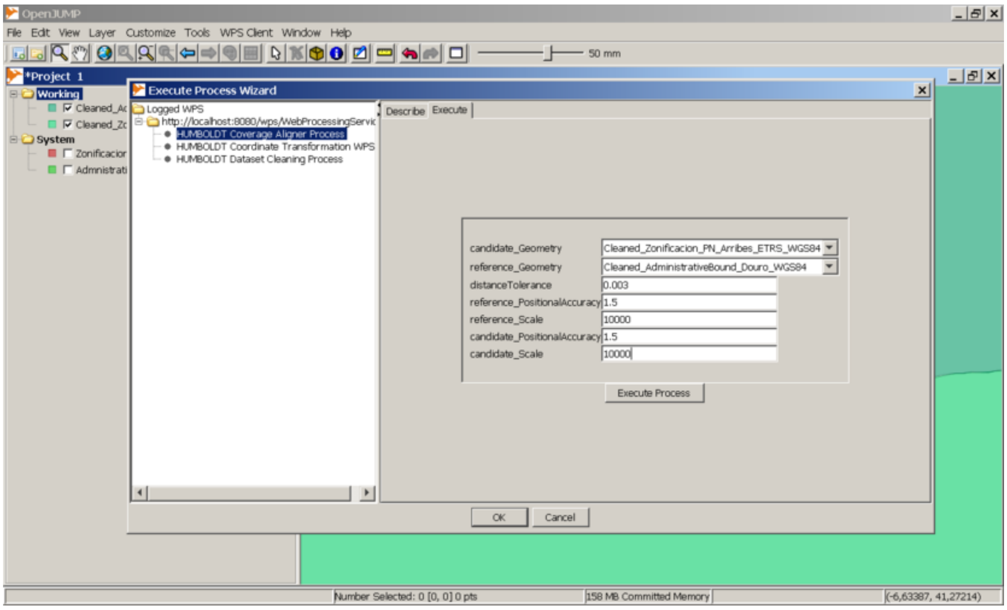
[3] `http://www.openjump.org`

[4] `http://udig.refractions.net`



(a) (b)

Figure 4: (**a**) Two lines to edge-match. The grey circles represent the threshold to apply. (**b**) The result of the edge-matching process where the erros are distributed: the new points are half-way between the points to snap. (The original lines are dashed)



OpenJUMP
File Edit View Layer Customize Tools WPS Client Window Help
50 mm
*Project 1
Working
☑ Cleaned_Ac
☑ Cleaned_Zc
System
□ Zonificacior
□ Administrati
Execute Process Wizard
Logged WPS
http://localhost:8080/wps/WebProcessingServic
HUMBOLDT Coverage Aligner Process
HUMBOLDT Coordinate Transformation WPS
HUMBOLDT Dataset Cleaning Process
Describe
Execute
Execute Process
OK
Cancel

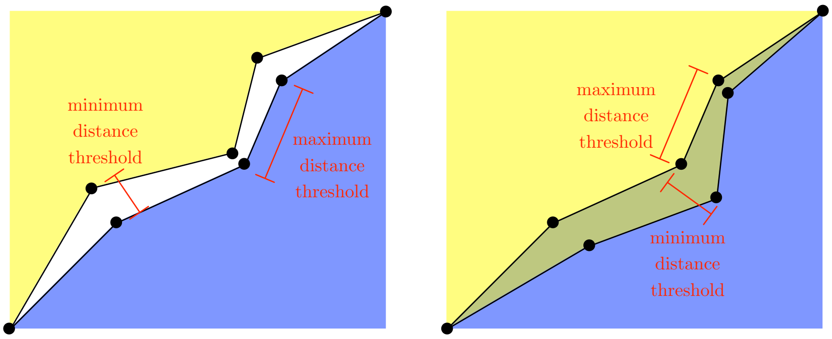
candidate_Geometry	Cleaned_Zonificacion_PN_Arribes_ETRS_WGS84
reference_Geometry	Cleaned_AdministrativeBound_Douro_WGS84
distanceTolerance	0.003
reference_PositionalAccuracy	1.5
reference_Scale	10000
candidate_PositionalAccuracy	1.5
candidate_Scale	10000

Number Selected: 0 [0, 0] 0 pts

158 MB Committed Memory

(-6,63387, 41,27214)

Figure 5: Example of the HUMBOLDT's EMS accessed from a plug-in of the open-source GIS OpenJUMP.



(a) Gaps between polygons

(b) Overlapping polygons

Figure 6: Defining a threshold for vertex and edge snapping. The threshold to use should be larger than the largest minimum distance between the matching boundaries, and smaller than the minimum distance between vertices of a single polygon.

3 Problems arising when edge-matching with snapping

The main problem of an edge-matching algorithm based on threshold/snapping lies in finding an appropriate threshold value for a given dataset. While in theory this value is linked to the accuracy of a dataset, in practice users do not always know how to translate the accuracy into a value, and if they choose the wrong value then their resulting dataset will not be properly edge-matched. Notice that while INSPIRE clearly states that each Thematic Working Group will define the appropriate threshold (INSPIRE, 2008), this is in our experience wishful thinking since the geographical datasets related to one theme usually come from different sources that have very different accuracies.

In brief, for a successful edge-matching based on snapping, here are some rules:

1. Adjacent polygons should not be further apart than this threshold along any part of their common boundaries (shown as the minimum threshold in Figure 6(a)). Otherwise, gaps are not able to be fixed.

2. Adjacent polygons should not overlap each other in areas which are further inwards than this threshold from their common boundaries (shown as the minimum threshold in Figure 6(b)). Otherwise, overlaps are not able to be fixed.

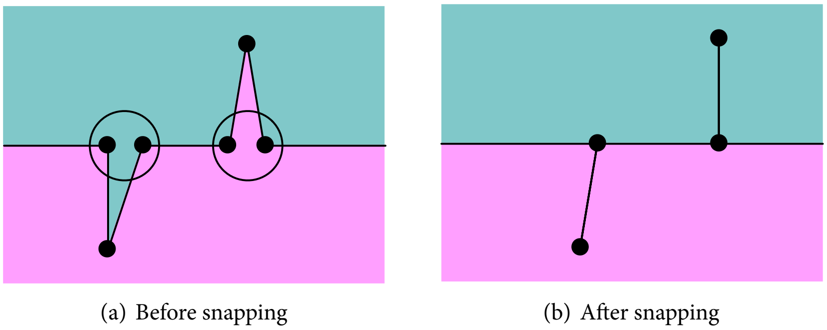
3. No vertices of a polygon should be closer to each other than this threshold, including non consecutive vertices (shown as the maximum thresholds in Figure 6). Otherwise, they might be snapped together, creating repeated vertices, disjoint regions, or various topological problems.

4. No vertices of a polygon should be closer than this threshold to any non incident edge. Otherwise, they might be snapped together, creating disjoint regions or various topological problems.

Furthermore, the threshold value is usually used for a complete dataset while the sizes of the gaps and overlaps between polygons might be different at different locations. What is worse is that sometimes such a “one-size-fits-all threshold” does not even exist (eg because point spacing might be in some places smaller than the width of the gaps and overlaps present); in Section 5 we present one such dataset.

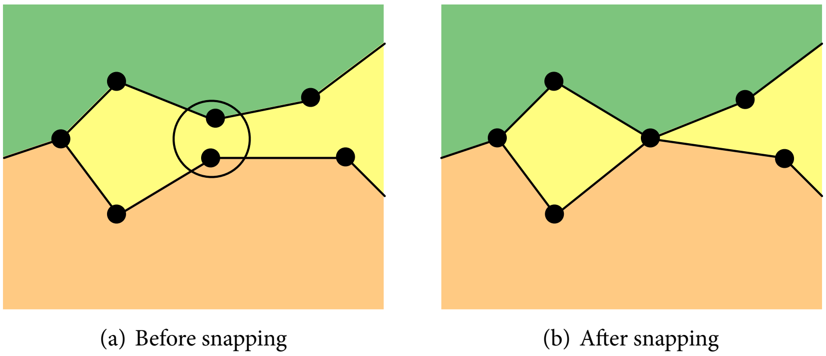
Even if the aforementioned conditions for a threshold are frequently not met (or are not checked beforehand), snapping is in practice still performed with a trial-and-error tolerance value. We highlight in this section the potential problems that snapping might create, ie the creation of invalid polygons and the changes in the topology of existing geometries.

Two examples are shown in Figures 7 and 8.


(a) Before snapping

(b) After snapping

Figure 7: Spikes and punctures can be created by snapping, since the bases of these elongated forms (encircled) might be narrower than the threshold, but their lengths not.


(a) Before snapping

(b) After snapping

Figure 8: Polygons can be split by snapping, since some parts might be narrower than the threshold (encircled). While this result does not create an invalid result, it can change the number of polygons present and their topological relations, and can therefore be undesirable.

While these examples prove that snapping is not problem-free, it should be said that commercial GIS packages often implement more complex snapping options (such as point-to-edge, edge-to-edge, or using a reference dataset). These options can help solve a problematic case, but can also complicate it by changing the topology of the polygons. One example is the post-processing operations to clean resulting polygons (eg disposing of polygons with small areas, removing redundant lines, thresholds for minimum angles, etc.) which might create new gaps and overlaps themselves, requiring an iterative cleaning process.

Another problem is that snapping is an intricate problem in itself, since there are many possible criteria that can be followed for both points and edges (eg points to the closest line, points to the closest point, points orthogonally to the closest line). Figure 9 illustrates one example where the resulting polygon is not valid anymore (and thus cannot be processed with a GIS).

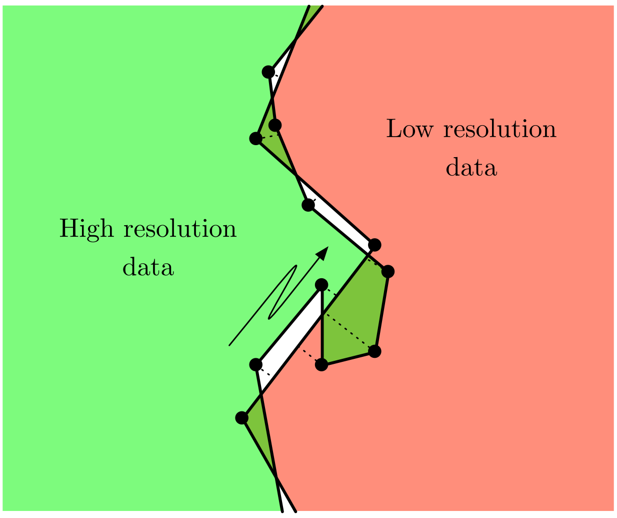


Figure 9: Snapping to the closest line can cause topologically invalid configurations. When two datasets of differing levels of detail are joined together by snapping the vertices of the high resolution dataset to the edges of the low resolution one, a situation where the line reverses on itself is created.

Finally, it is worth mentioning that although the edge-matching of two or more polygons *could* be done by snapping and splitting polygons, it might require the use of thresholds so large so as to have no physical basis, and result in polygons that are substantially different from the original data.

4 An alternative edge-matching algorithm based on constrained triangulations

Our approach to the edge-matching of polygons uses a constrained triangulation (CT) as a supporting structure because, as explained below, a CT permits us to fill the whole spatial extent of polygons with triangles, and then these allow us to identify easily the gaps and overlaps between different polygonal datasets. We use the idea of labelling each triangle with the label of the polygon

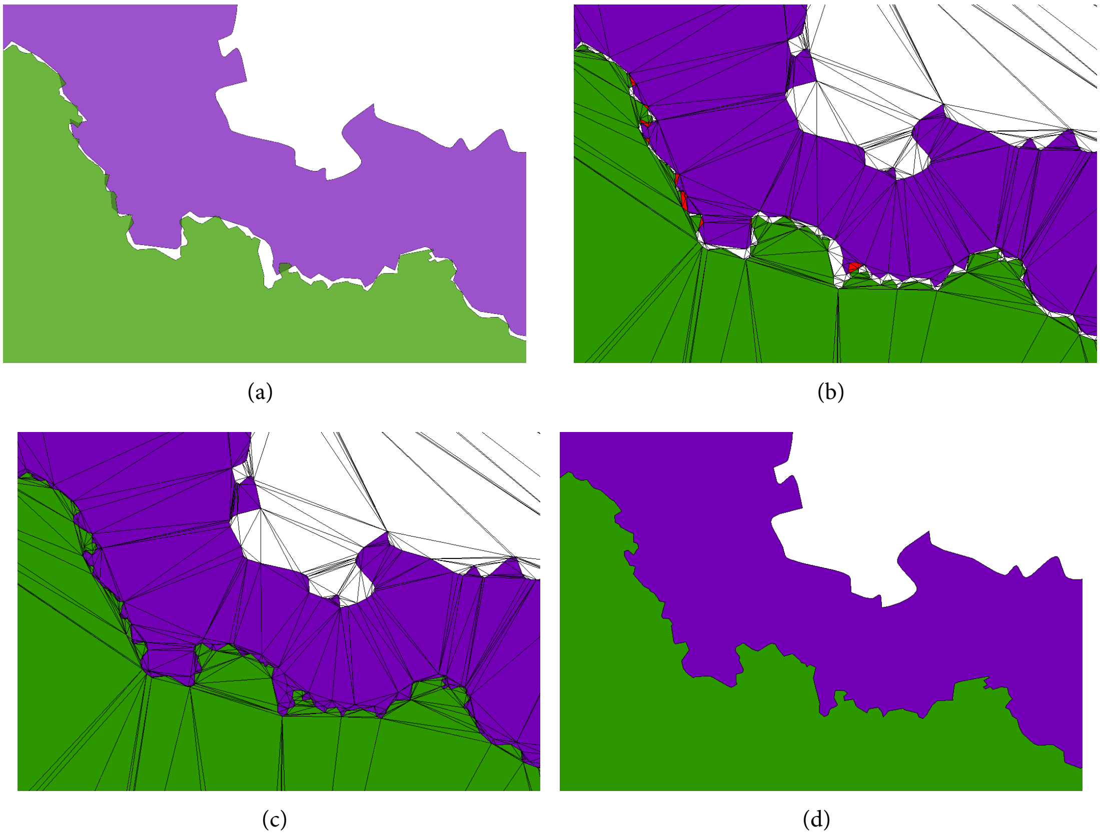


Figure 10: **(a)** Original dataset with two polygons. Notice the gaps (white) and the overlaps (darker green). **(b)** The CT of the input polygons; white triangles have no label, and red ones have > 1. **(c)** Triangles are re-tagged such that each triangle has one and only one label. **(d)** The resulting edge-matched polygons.

it decomposes: gaps will have no labels and regions where polygons overlaps will have more than one label.

The workflow of our approach is illustrated in Figure 10 and is as follows:

1. the CT of the input segments forming the polygons is constructed;
2. each triangle in the CT is labelled with the label of the polygon inside which it is located (see Figure 10(b));
3. problems are detected by identifying triangles with no label or more than one label, and by verifying the connectivity between the triangles;
4. gaps/overlaps are fixed locally with the most appropriate tag (see Figure 10(c));
5. edge-matched polygons are returned in a GIS format (eg a *shapefile*).

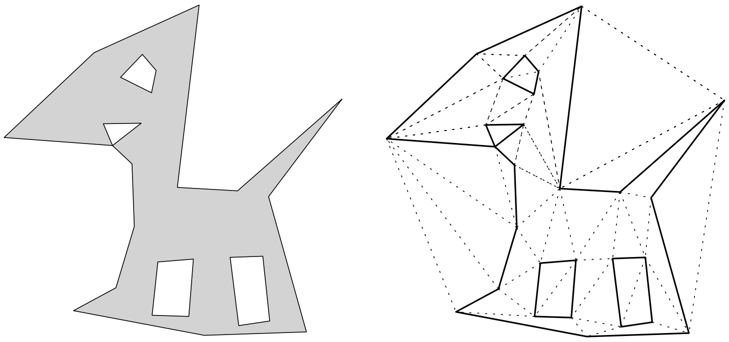


Figure 11: **(a)** A polygon with 4 holes. **(b)** The constrained triangulation of the segments of this polygon.

To construct the CT, tag the triangles, repair the problems and recover polygons, we use results we recently obtained for the validation and the automatic repair of planar partitions (such as the CORINE2000 land cover dataset). In Arroyo Ohori (2010) and Ledoux and Meijers (2010) we describe in detail the algorithms used to construct the CT of a set of polygons, to repair automatically planar partitions and to recover the polygons after the repair. We have modified slightly the algorithms and code so that we can perform the edge-matching of different polygons. We discuss below the main ideas, and we present in the next section some results.

Constrained triangulations. A constrained triangulation (CT) permits us to decompose an object (a polygon) into non-overlapping triangles, Figure 11 shows an example. Notice that no edges of the triangulation cross the constraints (the boundaries of the polygon). It is known that any polygon (also with holes) can be triangulated without adding extra vertices (de Berg et al., 2000; Shewchuk, 1997). In our approach, the triangulation is performed by constructing a CT of all the segments representing the boundaries (outer + inner) of each polygon. If two polygons are adjacent by one edge *e*, then *e* will be inserted twice. Doing this is usually not a problem for triangulation libraries because they ignore points and segments at the same location (as is the case with the solution we use, see Section 5). Likewise, when edges are found to intersect, they are split with a new vertex created at the intersection point.

Labelling triangles. The labels are assigned to the triangles by tagging the triangles adjacent to the edges of each polygon, and then visiting all the possible triangles with graph-based algorithms (ie depth-first search). See Arroyo Ohori (2010) for the details.

Identifying problems: gaps and overlaps. If the set of input polygons forms a planar partition, then all the triangles will be flagged with one and only one label. Problems (gaps and overlaps) are easily identified: all the triangles are visited and the ones having less or more than one label are returned.

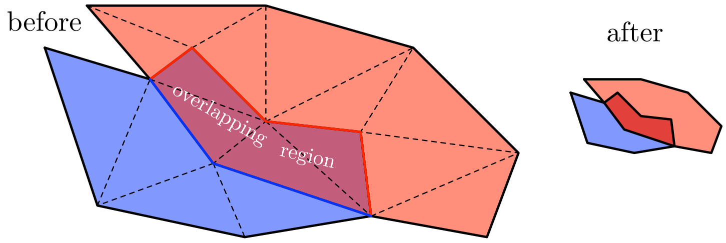


Figure 12: Regions are defined as adjacent triangles with equivalent sets of tags. In this example, the overlapping region between the red and blue polygons is repaired by the tag present along the longest part of the boundary surrounding the region (red).

Fixing problems: re-labelling triangles. Fixing a problem simply involves re-tagging triangles with an appropriate label. Arroyo Ohori (2010) proposes different repair operations that can be used to successfully fix gaps and overlaps. Four of them use triangles as a base (ie the label assigned is based on that of the 3 neighbouring triangles), which is faster and modifies the area of each input polygon the least. Two of them use regions of adjacent triangles with equivalent sets of tags (Figure 12), which is slower but yields results that are expected when edge-matching polygons.

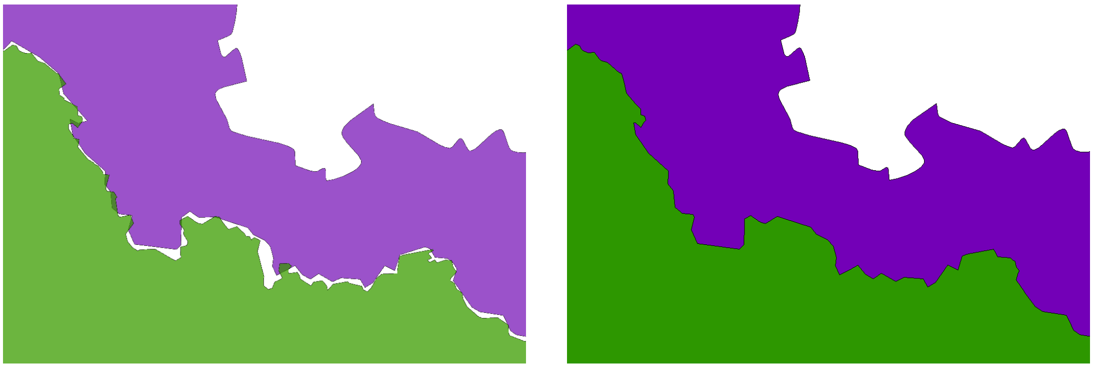
The most interesting repair operation for edge-matching is the one in which a *priority of labels* is used to repair regions, ie in case of gaps/overlaps the labels of adjacent polygons are ordered according to a user-defined priority, and the highest priority is assigned to the problematic triangles. We have adapted this operation so that the concept of *reference datasets* for edge-matching can be used. When a reference dataset is used, all the other datasets (we call them *slaves*) are snapped to it, and the reference dataset is not modified. When using a priority list, that means:

1. gaps should be filled with slave labels
2. overlaps should be fixed with the label of the master polygon.

Notice that in Figure 10(d) this technique was applied, and that the reference dataset (the green polygon) has not been modified. Figure 13 shows the result of edge-matching the polygons of Figure 10 with another criterion.

The main advantage of this approach is that the edge-matching can be performed with a *local* criteria, instead of a global one (the tolerance used is usually for the the whole dataset). It is also an efficient algorithm since only re-tagging triangles is involved to repair gaps and overlaps (which is a local operation).

Internal conflation. Observe that internal conflation, as described in the Introduction and in Section 2, is also elegantly performed with a triangle-based approach since the aim is to avoid gaps and overlaps within one dataset. Triangulating it and filling its holes with appropriate labels is easy, and moreover guarantees that valid geometries are returned.



(a)

(b)

Figure 13: **(a)** The same dataset as Figure 10(a). **(b)** Edge-matching performed with a repair operation where the label assigned to a problematic region is the one of the adjacent neighbour having the longest common boundary. Notice the differences with Figure 10(d).

Validation of results. If each triangle in the CT has one and only one label, then by definition there are no gaps and/or overlaps between triangles. Observe that triangles not located "between" polygons are ignored; they form the "universe", you can see some at the top-right of Figure 10(d) for instance. The greatest benefit of using a tagged triangulation for edge-matching polygons stems from the fact that while modification operations are performed, the validity of the polygons is always kept, together with the integrity of the data. This comes as a contrast to other methods, where care needs to be taken to ensure that the (geometric or topological) validity is not broken. For instance, if a zero width corridor that joins two regions is created, it should be detected and removed.

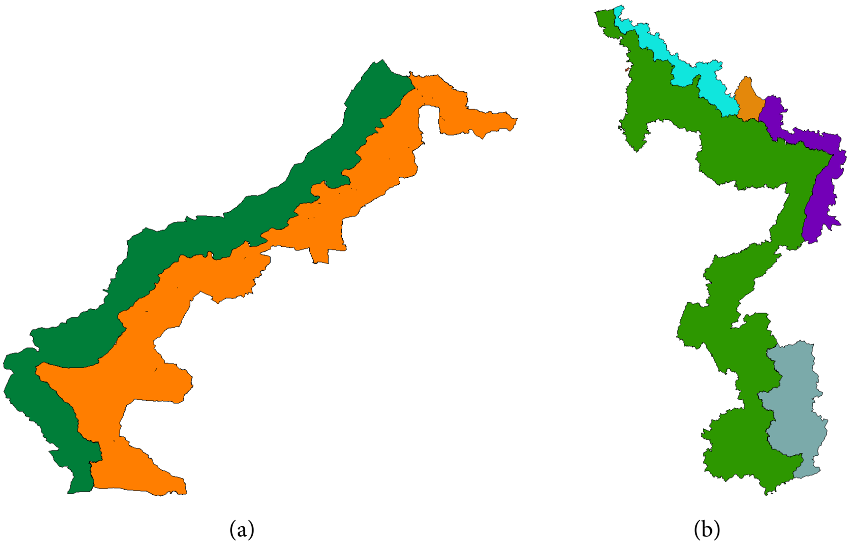
5 Experiments

We have implemented the algorithm described in this paper with the C++ programming language, using external libraries for some functionality: the OGR Simple Features Library, which allows input and output from a large variety of data formats common in GIS, and CGAL which has support for many robust spatial data structures and the operations based on them, including polygons and triangulations (Boissonnat et al., 2002). The developed prototype is open source and freely available[5].

We have tested our implementation with two datasets:

1. Figure 14(a): The border between Portugal and Spain along one national park is defined by a river. The Portuguese and the Spanish datasets do not match, see Figure 1 for one example at a larger scale. The two polygons have together about 12 000 points.

[5] On the GDMC website: `http://www.gdmc.nl`



(a)

(b)

Figure 14: **(a)** Border region between Portugal (green) and Spain (orange). **(b)** NUTS regions on the east of France (green), and some of its neighbouring countries (blue is Belgium; orange is Luxembourg; purple is Germany; grey is Italy).

2. Figure 14(b): The NUTS boundaries datasets of France and its neighbours. For France, we used the GEOFLA® dataset[6], and for Belgium, Luxembourg, Germany and Italy we used the dataset from UNEP/GRID-Geneva[7]. The larger-scale examples from Figures 10 and 13 are with these datasets. The polygons have together about 6 000 points.

As expected, we have been able to edge-match successfully these datasets, ie our output polygons were valid and no gaps/overlaps were present. Because we use an highly-optimised triangulation library, we could obtain results in about 0.3 s for the France dataset, and about 1 s for the Portugal-Spain dataset.

5.1 Comparison with other tools

As a comparison, we used FME[8] and the HUMBOLDT EMS to perform snapping.

FME could perform the matching with a given tolerance in about the same time (about 2 s), since it uses auxiliary data structures to speed up the process. EMS uses a *brute-force* implementation, where all the coordinates are compared with each other for snapping (thus 12 000 times 12 000 comparisons for the Portugal-Spain dataset; a quadratic behaviour), and took around 8 min to edge-match the Portugal-Spain dataset. It should be pointed out here that EMS is a Web-Processing

[6] Freely available from the website of the French IGN: `www.ign.fr`

[7] Available at `http://gcmd.nasa.gov/records/GCMD_GNV00159.html`

[8] `www.safe.com`

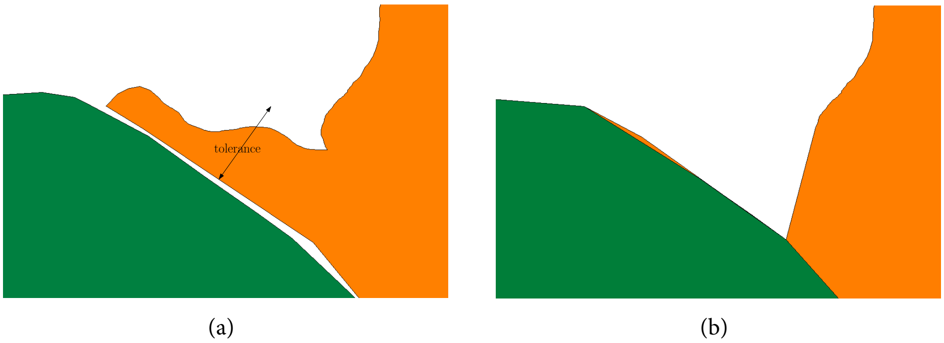


Figure 15: **(a)** Original dataset, with the tolerance used for snapping. **(b)** Collapsing of part of an polygon.

Service and that this time includes the conversion to GML and the uploading/downloading of the datasets to a server (we could not evaluate how much of the time was spent for these steps).

However, with both solutions, for both datasets, we could not find an appropriate tolerance with which valid geometries are produced and no gaps/overlaps remain. We applied a trial-and-error method, but as can be seen from Figure 10(a), the size of gaps and overlaps differ substantially. Some tolerance values could fix the gaps, but then other problems were created at different locations in the dataset. One such problem for the dataset Portugal-Spain is illustrated in Figure 15. To fix the gaps/overlaps, a large enough tolerance was needed, but this tolerance was also creating topological problems. Notice in Figure 15(b) that the area has been partially collapsed to a line because its width is smaller than the tolerance used; using a smaller tolerance solves that problem but creates others.

Since no snapping is used in the method we propose, such a problem cannot occur.

6 Conclusions

We have proposed a new algorithm to perform the edge-matching of polygons and we have shown that in practice it is highly efficient (since it is based on a highly optimised triangulator and only the labelling of triangles is involved), it avoids the pitfalls of choosing the appropriate threshold (if it even exists), and, perhaps more importantly, it guarantees that valid geometries are constructed, which permits practitioners to use the output for further analysis. Anyone who has tried—and perhaps failed—to find the appropriate threshold for a given dataset by using trial and error will recognise that our approach has great benefits.

However, it should be said that not everything is perfect, as Figure 16 illustrates. If two polygons do not touch or overlap, then the area connected to the universe will not be filled with labelled triangles and the resulting polygons will not be matched. These will happen at the "top" and the "bottom" of the edge-matching edge for two polygons. We are looking for a solution to this problem. One

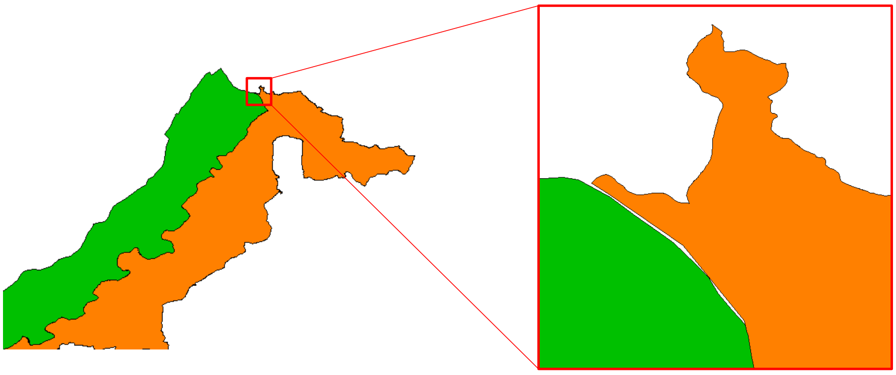


Figure 16: Same dataset as Figure 15, edge-matched with our approach. When polygons do not touch or overlaps, gaps can remain since these are considered part of the universe.

approach involves identifying thin or elongated triangles, and another involves snapping vertices as a pre-processing step to our approach (but since we use triangles afterwards, we should avoid the problematic cases, eg topological errors).

We plan in the future to add more repair functions, particularly one where we can edge-match two polygons without the notion of a master and a slave, ie as in Figure 4. Triangles can be used to find the centreline of a region, as Bader and Weibel (1998) showed. We also plan to modify the algorithm so that the edge-matching of lines is possible: these would be edge-matched, or "linked", with edges of the CT, although the use of a tolerance would still be necessary.

Acknowledgements

We would like to thank Roderic Molina for providing the Portugal-Spain dataset and, Jose Ignacio Gisbert for discussion about the implementation of the HUMBOLDT Edge Matching Service.

References

Arroyo Ohori K (2010). *Validation and automatic repair of planar partitions using a constrained triangulation*. MSc Geomatics, GIS technology group, Delft University of Technology, the Netherlands.

Bader M and Weibel R (1998). Detecting and resolving size and proximity conflicts in the generalization of polygonal maps. In *Proceedings 18th International Cartographic Conference*. Stockholm, Sweden.

Boissonnat JD, Devillers O, Pion S, Teillaud M, and Yvinec M (2002). Triangulations in CGAL. *Computational Geometry—Theory and Applications*, 22:5–19.

Burrough PA (1992). Are GIS data structures too simple minded? *Computers & Geosciences*, 18(4):395–400.

Davis M (n.a.). Java conflation suite. Technical report, Vivid Solutions. Available at `http://www.vividsolutions.com/jcs/`.

de Berg M, van Kreveld M, Overmars M, and Schwarzkopf O (2000). *Computational geometry: Algorithms and applications*. Springer-Verlag, Berlin, second edition.

INSPIRE (2008). Methodology for the development of data specifications. Annex I Data Specifications. Document D 2.6, version 3.0.

Ledoux H and Meijers M (2010). Validation of planar partitions using constrained triangulations. In *Proceedings Joint International Conference on Theory, Data Handling and Modelling in GeoSpatial Information Science*, pages 51–56. Hong Kong.

Lynch MP and Saalfeld AJ (1985). Conflation: Automated map compilation—a video game approach. In *Proceedings Auto-Carto VII*, pages 343–352.

Shewchuk JR (1997). *Delaunay Refinement Mesh Generation*. Ph.D. thesis, School of Computer Science, Carnegie Mellon University, Pittsburg, USA.

Yuan S and Tao C (1999). Development of conflation components. In *Proceedings of Geoinformatics*, pages 1–13. Ann Harbour, USA.