Solving the horizontal conflation problem with a constrained Delaunay triangulation

Hugo Ledoux and Ken Arroyo Ohori

3D geoinformation, Delft University of Technology

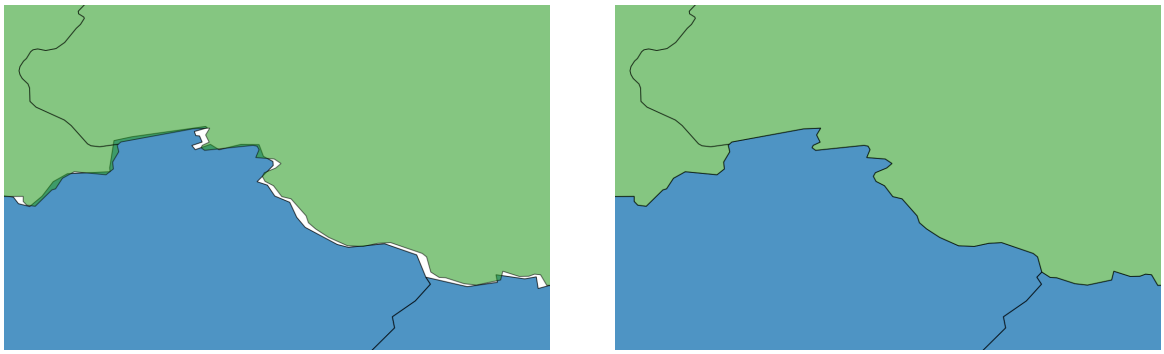
Hugo Ledoux and Ken Arroyo Ohori (xxxx). Solving the horizontal conflation problem with a constrained Delaunay triangulation. *Journal of Geographical Systems*, In Press.

The code of the project is available at
`https://github.com/tudelft3d/pprepair`

Datasets produced by different countries or organisations are seldom properly aligned and contain several discrepancies (eg gaps and overlaps). This problem has been so far almost exclusively tackled by snapping vertices based on a user-defined threshold. However, as we argue in this paper, this leads to invalid geometries, is error-prone, and leaves several discrepancies along the boundaries. We propose a novel algorithm to align the boundaries of adjacent datasets. It is based on a constrained Delaunay triangulation to identify and eliminate the discrepancies, and the alignment is performed without moving vertices with a snapping operator. This allows us to *guarantee* that the datasets have been properly conflated and that the polygons are geometrically valid. We present our algorithm, our implementation (based on the stable and fast triangulator in CGAL), and we show how it can be used it practice with different experiments with real-world datasets. Our experiments demonstrate that our approach is highly efficient and that it yields better results than snapping-based methods.

1 Introduction

One of the main issues when dealing with datasets produced by different countries or organisations is the management of the one-dimensional connections between geographical objects at their common boundaries (eg on either side of an international or provincial border). We need to ensure that the objects adjacent in reality are connected and that the discrepancies have been eliminated. These discrepancies are present because different equipments were used to collect the data, because the data are represented at different scales, because different coordinates reference systems are used and the conversion to a unified one introduces errors, or because the border is a natural feature such as a river [Ruiz et al, 2011]. This issue is often referred to as *horizontal conflation* since the datasets to be integrated do not overlap spatially (or only slightly, at their boundaries) [Yuan and Tao, 1999; Davis, 2003]. The other type of conflation is *vertical*:



(a) (b)

Figure 1: Two datasets: part of France (blue) and of Germany (green). Both datasets are formed of polygons representing the first administrative subdivisions. **(a)**: Original datasets (from different sources). **(b)** After horizontal conflation has been applied.

datasets covering the same area are combined, usually to create a new dataset whose accuracy is improved [Lynch and Saalfeld, 1985].

We focus in this paper on one aspect of horizontal conflation (see Figure 1): given two or more datasets composed of polygons (eg representing provinces, municipalities or cadastral parcels), we want to modify the boundaries of (some of) the polygons and "align" them so that gaps and overlapping areas are eliminated. As further explained in Section 2, this problem has been almost exclusively tackled by first isolating the boundaries of the datasets from the polygons they are incident to, and second using line-based algorithms to eliminate the discrepancies: corresponding real-world entities are matched using a threshold (a maximum searching distance), and then the alignment is performed by *moving* the lines to the same location (often referred to as "snapping"). We see several problems with this approach (we elaborate on these in Section 2). First, in our experience, finding a matching threshold that will resolve all discrepancies in a dataset is often impossible—a too small value prevents some lines to be matched, and a too large value creates invalid geometries somewhere else. Second, this problem is amplified if we consider that the edges are not in isolation but are the boundaries of polygons: the risk of collapsing parts of polygons to lines is high since it is tempting to use a large threshold to solve problems. Third, there are no reliable mechanisms to verify if the snapping was successful and what were the consequences of it on the geometries.

We present in Section 3 a novel algorithm to align the boundaries of adjacent datasets. Our algorithm differs significantly from other ones since it considers the polygon as its primitive (and does not isolate lines or focus solely on them), does *not* use a threshold to match features, and the alignment is performed without moving/snapping geometries. Instead, the discrepancies are identified and eliminated by using a *constrained Delaunay triangulation* (CDT) as a supporting structure. This allows us to *guarantee* that the datasets have been properly conflated and that the polygons are geometrically valid. We can furthermore indicate to the user what changes were made to the polygons.

The algorithm we present in this paper is an extension of our previous work for automatically repairing planar partitions: we filled small gaps and overlaps (often not visible to the user) by using a triangulation-based method [Arroyo Ohori et al, 2012]. Our extensions are threefold: (1) gaps and overlaps are eliminated in a manner this is consistent with how practitioners conflate datasets, ie by modifying first geometries with the lowest accuracy; (2) large problem areas (such as those in Figure 1a) are partitioned and each part is repaired independently, which ensures that the polygons involved to fix the area are not modify too much (the errors are distributed) and that the connection between features are preserved; (3) we propose a modification that allows us to perform what we call *spatial extent alignment*, eg to ensure that a set of polygons representing municipalities (or other administrative entities) fit exactly in a larger polygon representing their province. The latter improvements has the added benefits of allowing us to potentially align the boundaries of very large datasets, since we align them at different levels—from the neighbourhoods in a city to provinces in a country—and we can thus subdivide datasets into parts that fit in memory. We report in Section 4 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 different real-world datasets in Europe. We also demonstrate how boundary lines, often used in practice, can be incorporated in our approach. Finally, we discuss in Section 5 the shortcomings of our method and future work.

2 Related work

The specific issue we are addressing in this paper—the horizontal conflation of datasets formed by polygons—is rarely explicitly discussed in the scientific literature. The algorithms implemented in commercial software and used by practitioners for this case are the generic ones developed for either *edge-matching* or for vertical conflation.

Edge-matching. Edge-matching is a vague term that has no agreed-upon definition in geographical information systems (GIS). It is most often used to refer to linear features (eg roads or rivers) on both sides of a border (eg international or provincial) that need to be connected. Although its name implies otherwise, it usually means both finding the corresponding entities (matching) and aligning them (fixing the discrepancies). The matching between two features is performed by finding the closest pairs (of either vertex-vertex or vertex-edge), up to a given distance (a threshold, which is usually defined by the user). If two vertices/edges are matched, they are then *snapped* together so that they become the same object in the resulting dataset. Snapping is usually performed with one dataset being the *reference* dataset, that is, when matching, the geometries are moved to the ones in the reference dataset. The latter does not move because it is usually known that it is of higher accuracy. When the accuracy of both datasets is the same, or is not known, the errors can be distributed: two vertices to be snapped are then moved to their mid-point. The INSPIRE Directive mandates to use both methods (with and without reference dataset) for the harmonisation of European datasets [INSPIRE, 2008, Annex B].

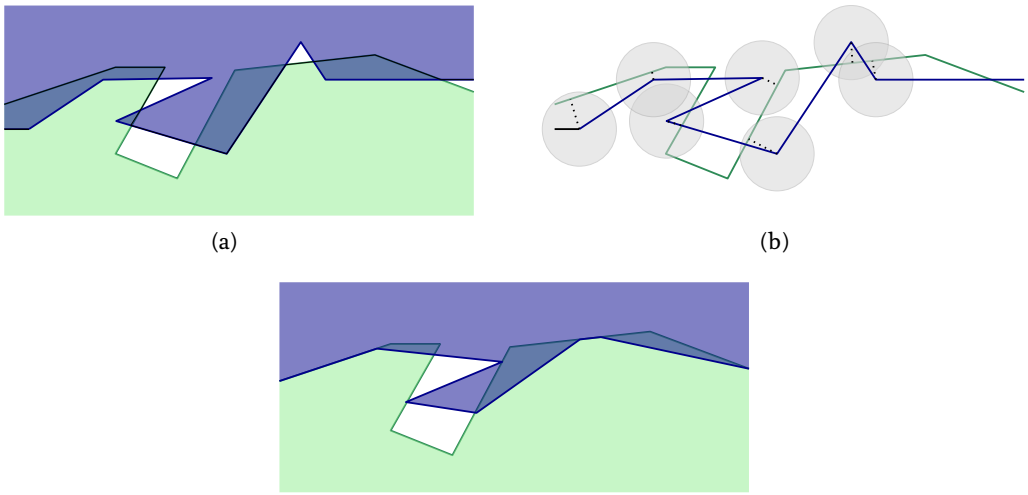
Most GIS packages implement edge-matching as described above (eg ArcGIS, FME, GRASS and Radius Topology), albeit the algorithms used differ in: (1) the order in which vertices are snapped together; and (2) what geometries are snapped (only to vertices or also to edges). It is also common in the scientific literature to see that snapping is being used with different types of datasets, eg cadastral boundaries [Siejka et al, 2013; Zygmunt et al, 2014], topographic datasets [Butenuth et al, 2007], digital gazetteer [Hastings, 2008], and census data [Schuurman et al, 2006].

As shown in Figure 2, if edge-matching is used for with polygons as input, then it implies that the vertices/edges on the boundary of each dataset are snapped to those in the adjacent dataset—the concept of lines being the boundaries of polygons is therefore ignored.

While snapping yields satisfactory results for simple cases, Arroyo Ohori et al [2012] demonstrate that for more complex ones it is often impossible or impractical to find a threshold applicable to the whole dataset, and that it is prone to errors that cause invalid geometries. There is often no single value that can be used for the whole dataset since it requires an almost uniform distribution of points, and discrepancies (gaps/overlaps) that are of the similar sizes. Observe the result in Figure 2c for instance, which snaps vertices from the top dataset to the closest geometry (vertex or line) in the reference dataset (bottom one). Even if other rules were used (eg snapping points only to other points), the results would still not be satisfactory. Also, the decision to snap or not parts of lines/polygons is often binary: all the vertices/edges closer than the threshold are snapped, the others not. While in theory the threshold 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 practice, we have observed that snapping is performed by practitioners with a trial-and-error threshold value. However, there exists no straightforward mechanism to detect automatically if the results are correct, ie if all discrepancies were resolved. Other tools have to be used (eg constructing the planar partition of the resulting datasets and comparing the number of polygons), and if there are problems then a different threshold value is used.

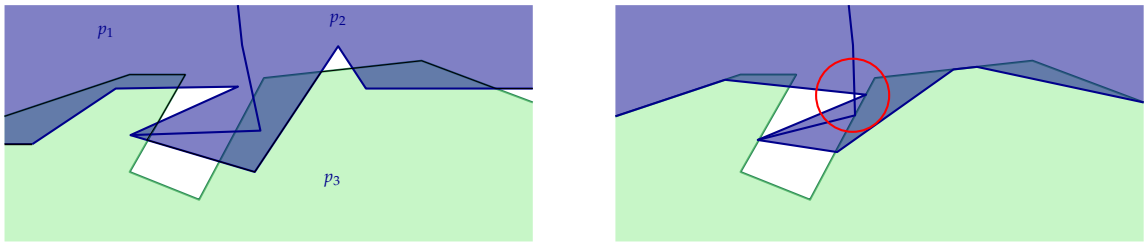
[1]The Computational Geometry Algorithms Library: `http://www.cgal.org`



(a) (b)

(c)

Figure 2: Edge-matching two datasets. **(a)** Input datasets (green is the master). **(b)** Snapping distance is shown as grey circles, and the closest geometry is shown by a dashed line. **(c)** Result of the edge-matching process.



(a) (b)

Figure 3: **(a)** One dataset to be conflated is formed by two polygons (p_1 and p_2). **(b)** Snapping the vertices of the blue dataset to the closest primitives in the green dataset introduces topological errors and intersections (in the red circle).

If we consider that in practice datasets very rarely contain only one polygon, snapping increases the risk of creating topologically invalid polygons and intersection polygons (because the boundaries between the polygons will also be snapped to the closest geometry); Figure 3 shows an example.

Klajnšek and Žalik [2005] consider, as is our case, datasets containing several polygons and find matching points and lines by using threshold distances. However, they do not align the boundaries, but instead simply removed from the dataset the matched geometries (since their aim is to create a dissolved dataset). The removals significantly simplifies the problem and avoid creating intersections.

Vertical conflation. The term vertical conflation was coined by Saalfeld in the early 1980s for the integration of two datasets representing the same region [Ruiz et al, 2011]. One dataset is always the reference (the master), and the other one (the slave) is aligned by first finding corresponding matching points (by using a threshold value) and then applying *rubber-sheeting*, ie a function to move all the other points according to the deviation of the control points. Gillman [1985] and Saalfeld [1988] use a triangulation to subdivide the slave dataset into subregions (triangles) and apply an affine transformation to each; the process is repeated iteratively until the displacement is minimal. Rubber-sheeting is meant to work with points and lines, and the concept of polygons (and their preservation) is not present. These could be reconstructed from the resulting lines, but there is n guarantee that the polygons will be preserved.

Beard and Chrisman [1988] use ideas of rubber-sheeting for the horizontal conflation of *map sheets*, ie datasets having only horizontal and vertical boundaries. The threshold to find matches is applied to points within a certain distance of the boundaries, and the lines incident to these are moved based on a rubber-sheeting function. To avoid moving all the points inside the dataset, a second threshold is used: only vertices closer to the boundaries are involved. There is no guarantee that after a movement the polygon or lines are valid (or that they are free of intersections).

Doytsher [2000] tackles the exact same problem as we do by first matching vertices (also using a tolerance) and then moving *all* the vertices in the dataset according to a rubber-sheeting transformation. The RMS errors of the transformation are minimised, and while it is claimed that the topological properties of the polygons are preserved, it is not proved and we do not see how this could be the case (since there are no constraints when moving vertices).

3 Aligning the boundaries of datasets with a triangulation-based algorithm

Our approach to aligning the boundaries of polygons builds upon our previous work to automatically repair the small gaps and overlaps in planar partitions where all polygons are stored independently (with the Simple Features paradigm [OGC, 2006], a *shapefile* for instance). In Arroyo Ohori et al [2012], we used a constrained triangulation (CT) as a supporting structure because, as explained below, it permits us to fill the whole spatial extent of the datasets with triangles, and then the triangles 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 it decomposes: gaps will have no labels and regions where polygons overlap will have more than one label. Repairing implies relabelling triangles so that each triangle has one and only one triangle.

We first briefly describe in Section 3.1 the original algorithm for repairing planar partitions. Then, in Section 3.2, we describe our extensions to this algorithm so that the boundaries of datasets can be properly aligned when the datasets are formed by several polygons and when the gaps/overlaps are large. We also propose two modifications to our approach (in Sections 3.3 and 3.4) so that two common conflation issues faced by practitioners can be solved: how to perform horizontal conflation of polygons against a boundary represented as a line, and spatial extent conflation.

3.1 Using a CT to repair a planar partition

The workflow of Arroyo Ohori et al [2012] is illustrated in Figure 4 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 identifier of the polygon inside which it is located;
3. problems are detected by identifying triangles with no label or with two or more labels;
4. gaps/overlaps are fixed *locally* with the most appropriate label;
5. modified polygons are reconstructed and returned in a GIS format (eg a *shapefile*).

Constrained triangulations. A constrained triangulation (CT) permits us to decompose one or more polygons into non-overlapping triangles, Figure 5 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 original approach, the triangulation was performed by constructing a CT of all the segments representing the boundaries (exterior and interior) of each polygon.

If, as in Figure 5, 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 4). Likewise, when edges are found to intersect, they are split with a new vertex created at the intersection point. These are the only vertices that are added during the conflation process.

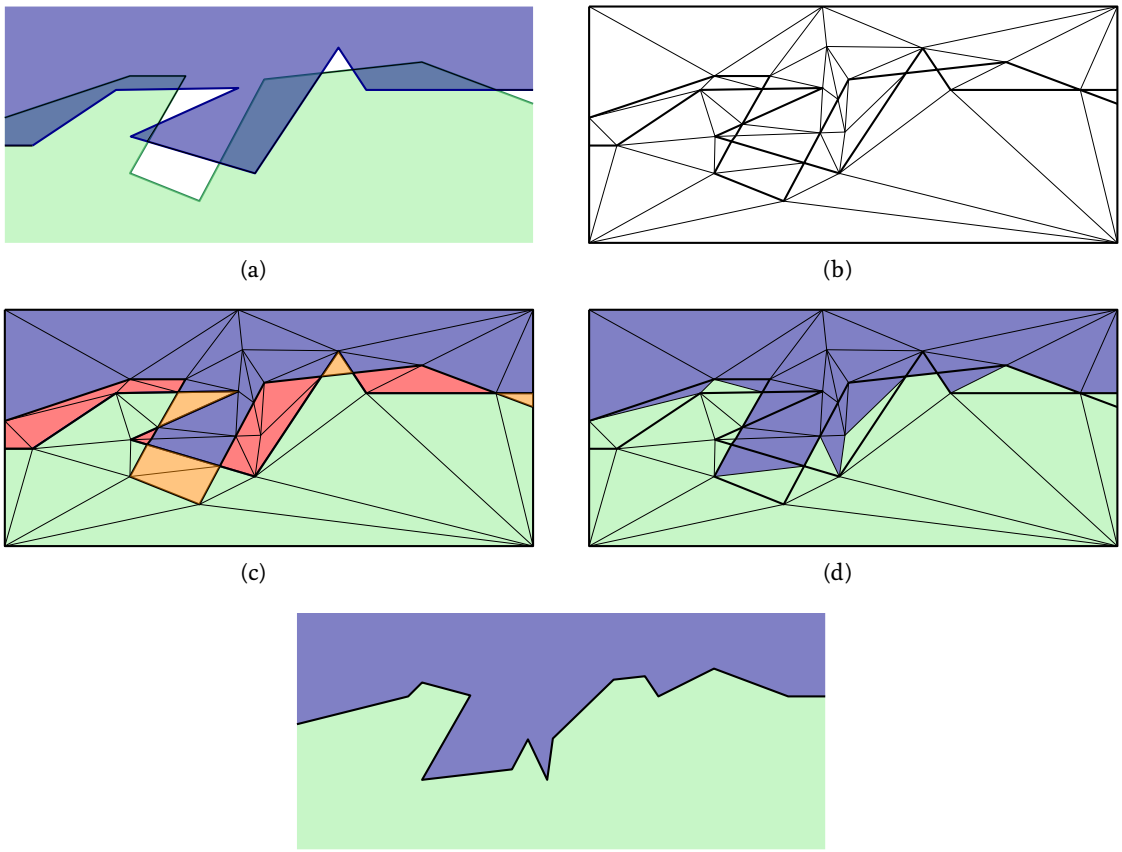


Figure 4: **(a)** Original dataset with two polygons (same as Figure 2). **(b)** The CT of the input polygons. **(c)** Labelling of the triangles: the colours represent the label of each polygon (orange = no label; red = 2+ labels). **(d)** Triangles are re-labelled such that each triangle has one and only one label. **(e)** The resulting alignment of the two polygons.

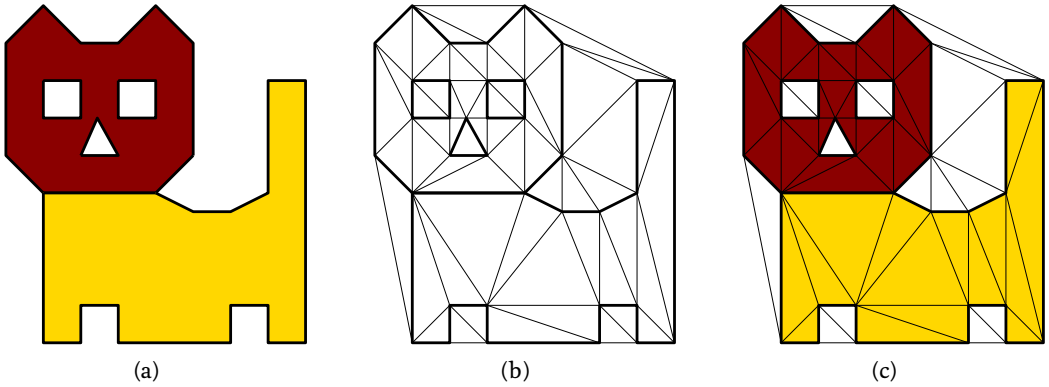
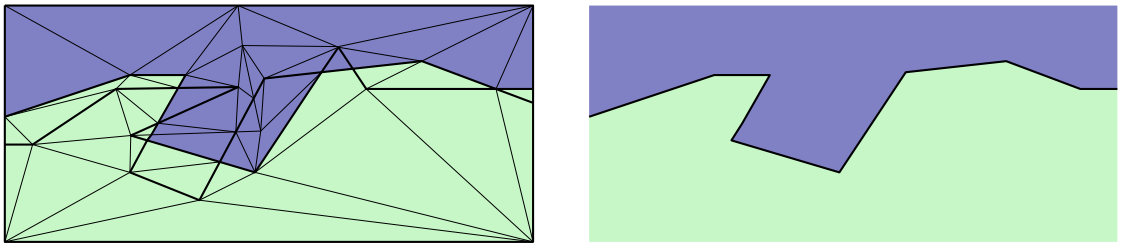


Figure 5: **(a)** Two adjacent polygons, one of them containing 3 holes. **(b)** The constrained Delaunay triangulation of the segments of these two polygons. **(c)** From the CT, it is possible to obtain the triangles decomposing each polygon (here represented with the colour).



(a) (b)

Figure 6: Dataset from Figure 4a when a region-based repair method is used (based on longest-edge). **(a)** Labelling of the triangles. **(b)** The resulting alignment of the two polygons.

Labelling triangles. The labels are assigned to the triangles by first labelling the triangles adjacent to the edges of each polygon, and then visiting all the triangles with graph-based algorithms (ie depth-first search) without traversing constrained edges (the original boundaries of the input polygons). Triangles located inside the convex hull of the dataset, but not decomposing any polygons, are labelled with a special label 'universe'. See Arroyo Ohori et al [2012] for the details.

Identifying problems: gaps and overlaps. If the set of input polygons forms a planar partition, then every triangle will be labelled with one and only one label. Problems are easily identified: gaps are formed of triangles having no label, and overlaps of triangles having two or more labels.

Repairing problems: relabelling triangles. Repairing a gap or an overlap simply involves relabelling the triangles with an appropriate label, which means that the label assigned should be the same as one of the 3 neighbours, otherwise regions can become disconnected. Arroyo Ohori et al [2012] proposes 6 repair operations. Four of them use triangles as a base: the label assigned is based on that of the 3 neighbouring triangles, for example the label present in the largest number of adjacent triangles is assigned (this method is used in Figure 4e). Triangle-based operators are fast (purely local operations that are performed in constant time) and modify the area of each input polygon the least. However, the shape of the resulting polygons can be significantly different from the original (because of the 'spikes' that are created).

The other two methods use regions of adjacent triangles with equivalent sets of labels, which is slower but generally yields results in which the polygons have less spikes. Figure 6 shows one example where each of the eight problematic regions of Figure 4a is assigned one label. The label is obtained with the *longest-boundary* method, ie the boundary of the problem region is first decomposed according to the label of the triangles incident to it (but outside the problem region), and second the label is that of the longest portion of the boundary.

Notice that these repair operations can be used one after the other, for instance if first the repair according to the largest number of adjacent triangles has a tie, then this is solved by using another method (or randomly choose one)—this was used in Figure 4e.

3.2 The extensions necessary for horizontal conflation of polygons

To align boundaries of polygons, we extend and modify the repair algorithm described above. The extended algorithm to align the boundaries of two or more datasets is described in Algorithm 1 and contains three improvements: (1) priority list of datasets; (2) gaps and overlaps treated differently; (3) subdivision of gap regions by adding extra constrained edges.

Priority list of datasets. The first extension is that we use a new operator based on a *priority list of datasets*. It is a generalisation of the concept of a reference dataset often used for edge-matching. We extend this concept so that several datasets can be used all at once (instead of performing edge-matching with only two datasets). The input of the algorithm has a priority list with all the datasets involved, ordered based a given criterion. The criterion is usually the accuracy of the datasets, but others can be used. In a given problem region between two adjacent datasets, the one with the lowest accuracy (priority) should be moved/modified. The first dataset in the list is

Algorithm 1: AlignBoundaries

Input: An ordered list L of datasets (acc. to a given priority)
Output: the boundaries of the datasets in L are aligned

```
1  T ← CDT of segments in L;
2  label each triangle in T with (dsid, pid);
   // dsid = input dataset; pid = ID of the polygon
3  for each overlap region do
4      l ← label whose dsid is highest in L;
5      relabel triangles with l;
6  for each gap region do
7      ExtraConstraints (Algorithm 2);
8  while some triangles in T have 0 label do
9      for each gap subregion do
10         if subregion adjacent to > 1 datasets then
11             l ← label whose dsid is lowest in L;
12             relabel triangles with l;
```

the master and others are its slaves; for instance, referring to Figure 7, the list would be $[M, T, S]$, since we assume that M has the higher accuracy, and S the lowest.

Gaps and overlaps handled differently. The gap and overlap regions are handled differently, and this allows us to modify the boundaries of datasets in a way that is consistent with the master-slave paradigm. A second modification is that the labels assigned to each triangle are formed by a tuple of the dataset and the unique identifier of the polygon: $(dsid, pid)$. For an overlap region, the label used to relabel all triangles is that whose *dsid* is the highest in the priority list, eg in Figure 7c the 3 overlapping regions (red regions) are filled with polygons from the dataset M (having labels m_1, m_2 and m_3). For a gap, this label is the lowest in the priority list; the candidate labels are those adjacent to the gap region. If a gap region is adjacent to more than 1 polygon from the same dataset (see for instance Figure 7b where both s_1 and s_2 could be assigned to the region), then the ID can be determined with the one of the repair methods mentioned above, such as the longest boundary.

However, aligning boundaries with this approach gives unsatisfactory results for gaps, that is we obtain results that are far from what a human being would manually do. For instance, in Figure 7, the gap between the datasets is entirely assigned to the polygon s_2, while we could argue that some parts should be assigned to s_1 and t_1. Observe that the overlap regions do not suffer from this problem since the constrained edges divide the region; in Figure 7c the top overlap region between M and S is divided into two sub-regions by the constraints, and each gets the appropriate label (m_1 and m_2).

Subdividing gap regions. To improve the labels assigned to gaps, we propose a heuristic to subdivide the gap regions into subregions. This is achieved by inserting extra constrained edges in the triangulation (as shown in Figure 8). The constrained edge are not new edges, but rather existing edges that are labelled as constraints (thus no complex geometric operations are involved). Observe that while a generic constrained triangulation was sufficient to perform validation and repair in our original work [Arroyo Ohori et al, 2012], here a constrained *Delaunay* triangulations (CDT) is required. A CDT is a triangulation for which the triangles are as equilateral as possible [Chew, 1987]. We use the edges of the triangles (and their lengths), and having well-shaped triangles is an advantage.

For a given gap region, we proceed as follows (see Algorithm 2). First, we visit each vertex v on the boundary of the gap (there are 9 in Figure 8a) and identify *split vertices*: vertices whose number of incident constrained edges in the CDT is more than 2. These allow us to identify where different polygons of the same dataset are adjacent (eg vertices a and d in Figure 8a) and also where different datasets are adjacent (eg vertices b, c and e in Figure 8a).

Figure 7: **(a)** Three datasets (S=purple; T=grey; M=green), having respectively 2, 1 and 3 polygons, need to be aligned. **(b)** The CDT of the datasets with problematic regions highlighted (red=overlaps; orange=gaps). **(c)** Overlaps are repaired with the random neighbour method. **(d)** Holes with the same method. **(e)** Resulting aligned datasets.

Algorithm 2: ExtraConstraints

Input: A gap region R in a labelled CDT $\mathscr{T}$; and a maximum length l
Output: $\mathscr{T}$ with potentially some extra constrained edges added in R

```
// let S be the list of split vertices
for each vertex v ∈ R do
    if degree(v) > 2 // only constrained edges are considered
    then
        add v to S;
for each s in S do
    e ← get shortest edge su (where u ∈ S);
    if (e ≠ NULL) and (length(e) < l) then
        insert constraint e in 𝒯;
    else
        e ← get shortest edge su (where u ∈ R);
        if (e ≠ NULL) and (length(e) < l) then
            insert constraint e in 𝒯;
```

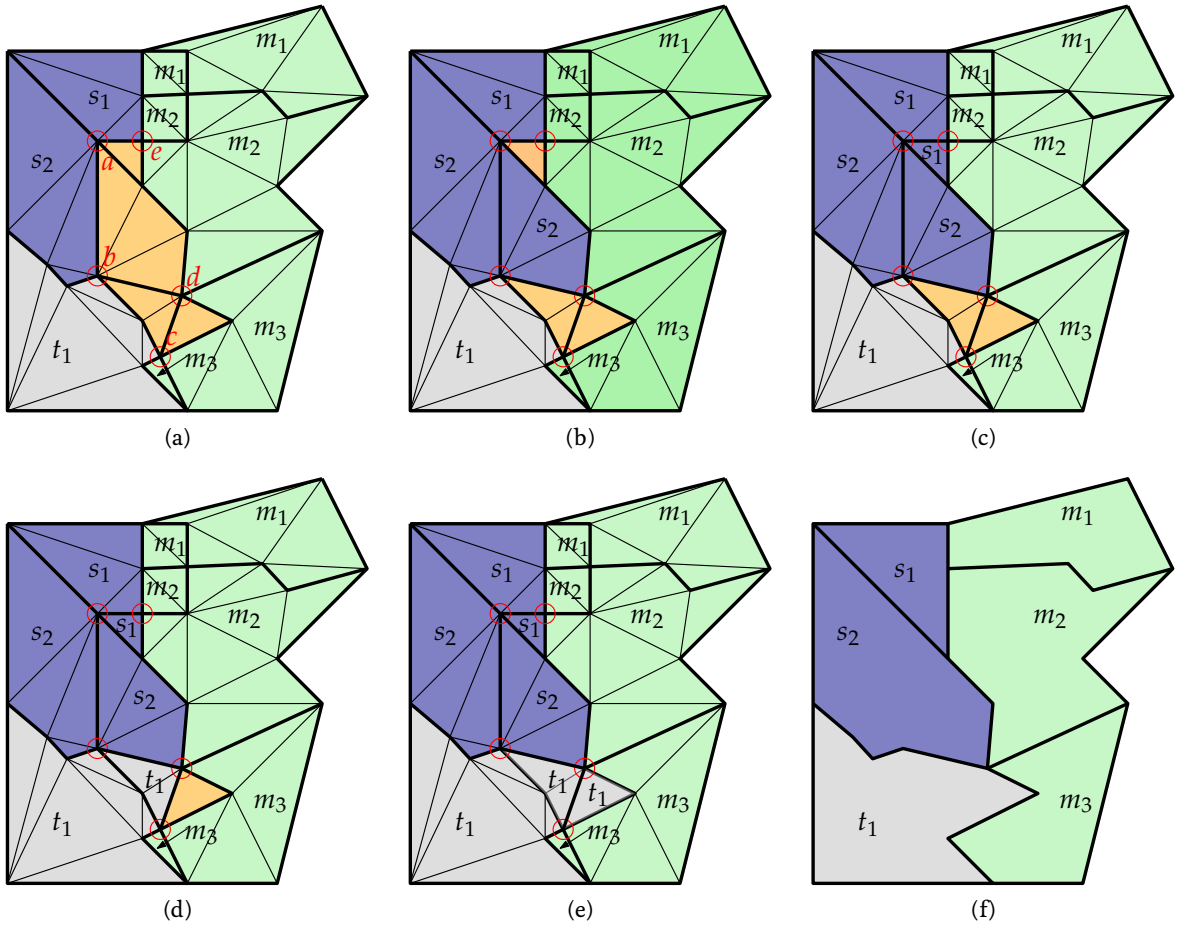


Figure 8: The same three input datasets as in Figure 7. The overlapping regions have already been repaired. **(a)** 5 split vertices are identified (a, b, c, d, and e), and 3 extra constrained edges are inserted in the CDT (bd, cd and the one incident to a), separating the gap region into 4 sub-regions. **(b)** The sub-region adjacent to s_2 is relabelled. **(c)** The sub-region adjacent to s_1 is relabelled. **(d)** The sub-region adjacent to t_1 is relabelled. **(e)** The last sub-region is relabelled (to t_1). **(f)** The resulting aligned datasets.

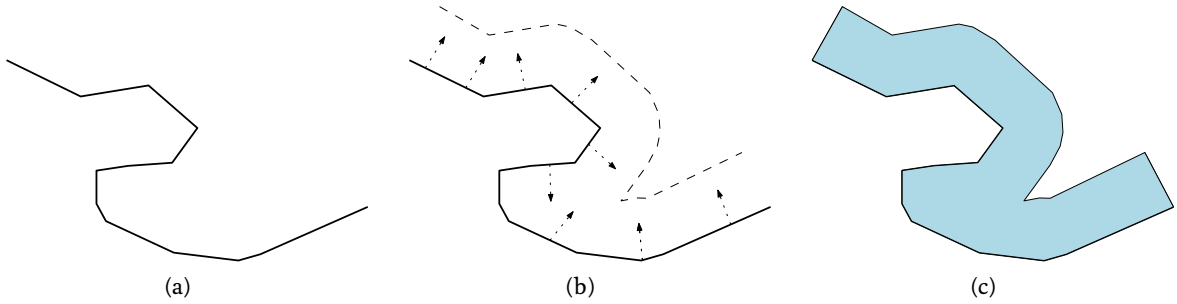


Figure 9: Creation of a polygon from **(a)** an input line that is **(b)** offset by a given distance. **(c)** The resulting polygon

Second, for each split vertex, we try to insert one constrained edge inside the gap region (which means here that the edge is not on the boundary of the region). An edge whose both ends are split vertices is favoured, if there are none then the shortest edge is inserted as a constraint. Only edges that are shorter than a user-defined maximum length are inserted as constraints. It is possible that for a candidate vertex no constraint is inserted, either because there is no incident edges inside the gap region (e in Figure 8a), or that the constraint is already present. In Figure 8a, the extra constraints bd is added twice; the second insertion does not modify the CDT.

The algorithm for labelling gap regions is then used as described in Section 3.1, but now each subregion is processed separately; after the insertion of the extra constraint in Figure 8a there are thus 4 gap regions. In the Algorithm 1, the relabelling of regions are not applied to the CDT directly (since the result of one could influence another, creating dependence on the order in which the triangles are visited). Instead, triangles that have been relabelled are saved in a separate list, and only after all the triangles have been visited are the new labels applied. Because the insertion of extra constraints isolates subregions, not all subregions can be directly relabelled. A subregion can be relabelled only if it is adjacent to two or more datasets. If a subregion is adjacent to only one (eg the lower-right orange region in Figure 8a), then it is not repaired until one of its adjacent is relabelled (Figure 8d). This implies that several 'passes' over all the triangles have to be performed, each pass tries to find a new label for triangles having 0 label and these are applied at the end of the pass.

Our heuristic has the added benefit of connecting lines that are the boundaries of polygons and/or datasets and of preserving better the area of polygons since gap regions are split and subregions are assigned to different polygons. We demonstrate in Section 4 the results we obtained with real-world datasets.

3.3 Edge-matching to a linear boundary

In practice, international and delimiting boundaries between countries and administrative entities are often available as *lines*, because these are agreed upon by the neighbouring parties. A trivial modification of our algorithm can be made for such cases: the linear boundary is converted to a polygon. As shown in Figure 9, this can be easily done by first offsetting the line by a distance d (to either side of the line), and then linking the two lines (this can be seen as a 'half-buffer'). We demonstrate in Section 4 how this performs with real-world datasets.

3.4 Spatial extent conflation

Another minor modification to Algorithm 1 allows us to perform what we refer to as spatial extent conflation. This is used to ensure that a set of polygons fits exactly in a spatial extent polygon that represents the entity higher in the hierarchy for instance. It is a common problem for practitioners who deal with administrative and territorial units, among others[2]. Each country is divided into administrative units at different administrative levels in a hierarchy. In the

[2]These units are called in Europe 'Nomenclature of Territorial Units for Statistics', or NUTS

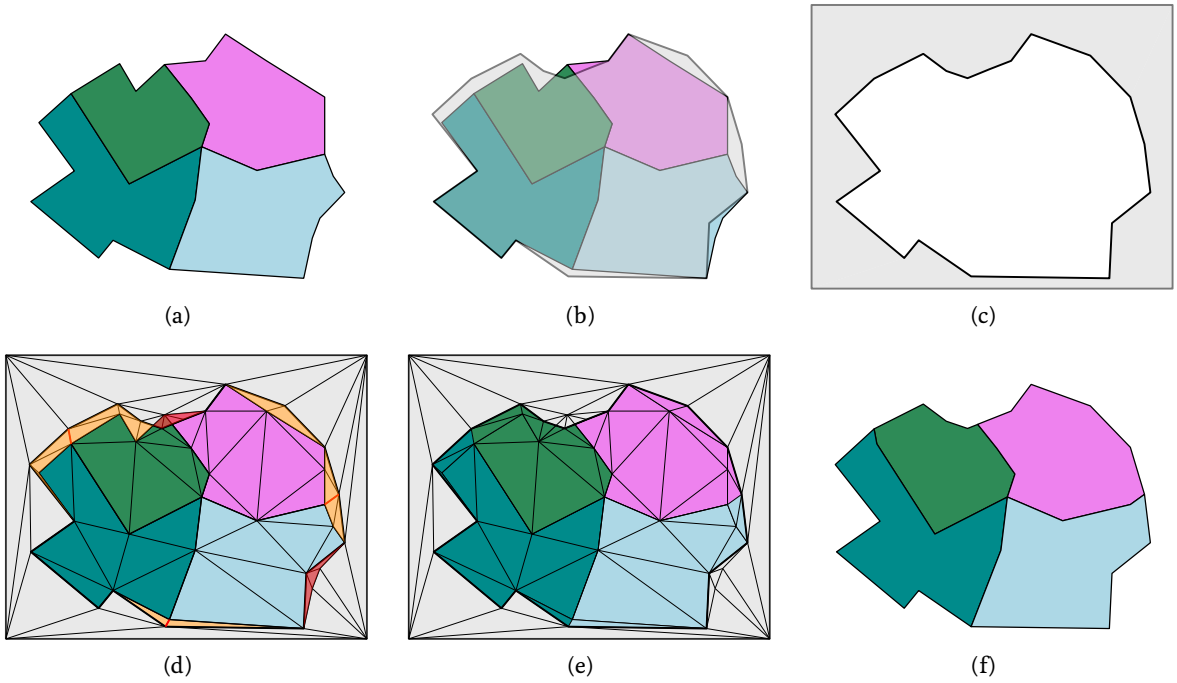


Figure 10: **(a)** Four polygons with **(b)** the spatial extent polygon to which they should be aligned. **(c)** The spatial extent polygon becomes the hole of a larger polygon. **(d)** CDT + labelling + insertion of extra constraints. **(e)** Relabelling of the problem regions. **(f)** Resulting aligned datasets.

INSPIRE directive[3], the hierarchy goes from the national level to the 6th level (sub-city level in most cases) [INSPIRE, 2014]. All the units at one level should fit exactly inside its parent unit, and there should not be any gaps.

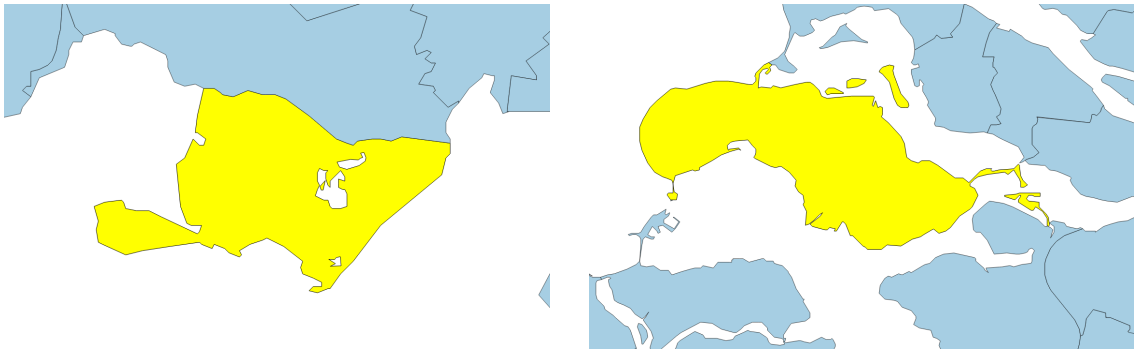
Figure 10 shows the main modification to our algorithm: the polygon representing the spatial extent becomes a hole of a larger polygon (whose shape is irrelevant, it can be obtained by enlarging the axis-aligned bounding box of all the polygons by 10% for instance).

The only modification necessary to Algorithm 1 is that a new label, let us call it `spatialextent`, is assigned to the triangles decomposing this polygon and it is assigned the highest priority. Triangles having this label are ignored during the process of reconstruction of the polygons.

In practice, the spatial extent polygon is not always a single polygon. As Figure 11 shows, municipalities or counties are often disconnected (eg due to islands) and have inner rings (holes). Thus, the creation of the polygons used for the input can be obtained by first generating a large rectangle containing all the spatial extent polygons, and then doing a Boolean difference between the two.

Observe that aligning polygons to a spatial extent allows us to align the boundaries of very large hierarchical datasets since we can align them individually at each level. As shown in Figure 12, if we begin the alignment at the highest level (the 1st level being the spatial extent of the polygons of the 2nd level), and then continue iteratively to the lower administrative level using the previously generated results (ie in Figure 12 the 7 polygons are used as spatial extents for the 3rd level, and so on), then the memory footprint of the alignment should remain relatively small. That is, the process is bounded by the size of the largest polygon and its decomposition one level lower. We demonstrate in the next section an example with real-world examples.

[3]The European directive that ensures that the spatial data infrastructures of the Member States are compatible with each others and usable.



(a) (b)

Figure 11: Examples of two municipalities in the Netherlands (in yellow) having disconnected regions. **(a)** Baarle-Nassau has 3 Belgian enclaves (white holes), and one of them has an enclave belonging to Baarle-Nassau. **(b)** A municipality formed by 5 different islands (Schouwen-Duiveland).

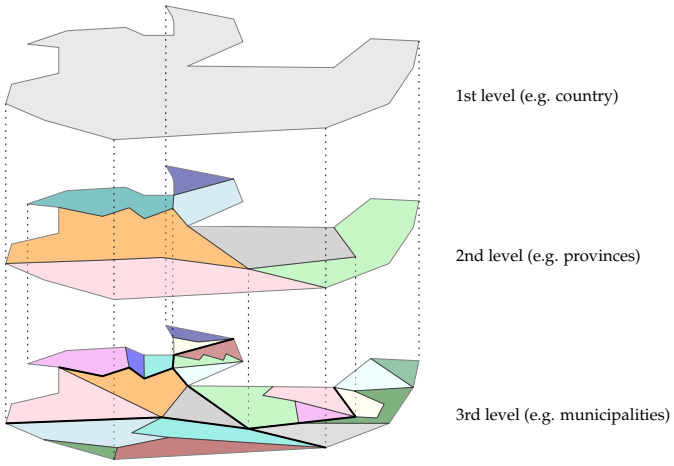
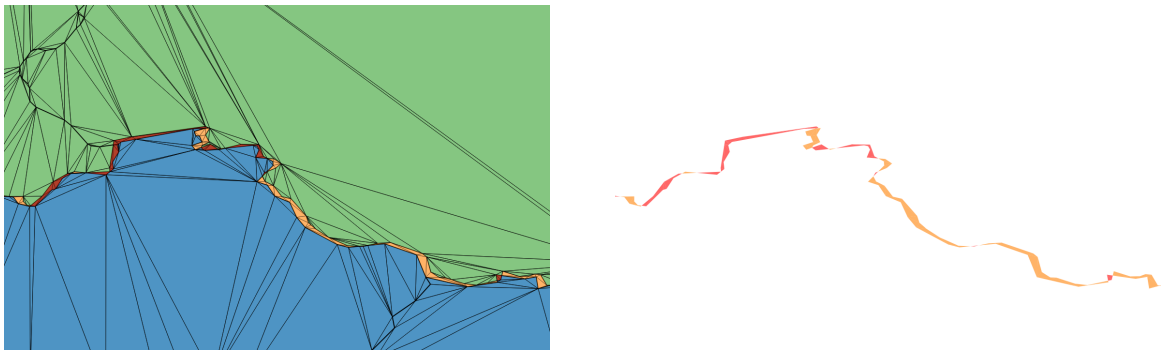


Figure 12: Top 3 levels of a NUTS dataset.



(a) (b)

Figure 13: For the datasets in Figure 1a, **(a)** labelled triangulation of the datasets, and **(b)** errors as polygons. For both: red=overlap and orange=gap.

4 Implementation & Experiments

We have implemented the algorithm described in this paper with the C++ programming language, using external libraries for some functionalities: (1) GDAL[4], which allows us to read/write from/to a large variety of data formats common in GIS and to handle the attributes of each polygon; and (2) CGAL[5] which has support for many robust spatial data structures and the operations based on them, including polygons and triangulations [Boissonnat et al, 2002].

Our prototype is open-source (under a GPL license) and can be freely downloaded at `www.github.com/tudelft3d/pprepair`. It can read most GIS formats, perform the alignment of the boundaries, and return to the user the input files with the boundaries modified; all the attributes of the polygons are preserved. Furthermore, as shown in Figure 13, the user obtains an overview of which polygons were modified, and the modifications are also exported as a GIS file (in the form of polygons showing parts that were added/removed from the datasets). This is something practitioners often mention as useful ("how much was my dataset modified?") and that is rather difficult to accomplish with traditional snapping-based alignment methods. The only solution is to perform a Boolean difference between the input dataset and the output, which is computationally expensive and prone to floating-point errors.

We have tested our implementation with several real-world datasets that are publicly available, and we report on some of these in the following. Our experiments cover a broad range of alignment problems that practitioners have to solve. Furthermore, our implementation handles `MultiPolygons` and polygons with holes, which are very common in practice when dealing with administrative subdivisions and other datasets related to the territory.

4.1 Experiment #1: Conflation of polygons

The eastern border of France was selected as a test area, and subsets of these two datasets are used (shown in Figure 14):

1. the GEOFLA[6] of the 2nd level of the administrative units in France. [21 polygons – 6,678 vertices]
2. The Vector Map (VMAP) at the lowest resolution (VMAP0)[7]. [37 polygons – 10,256 vertices]

To obtain a comparison with other tools, we performed snapping as described in Section 2 with FME[8]. We used the `AnchoredSnapper` transformer with the VMAP0 as master dataset; vertices in the slave dataset (GEOFLA) are snapped to the closest vertex/line in the VMAP0 dataset (a rather

[4] Geospatial Data Abstraction Library: `www.gdal.org`
[5] Computational Geometry Algorithms Library: `www.cgal.org`
[6] `http://professionnels.ign.fr/geofla`
[7] `http://gis-lab.info/qa/vmap0-eng.html`
[8] `http://www.safe.com/fme/fme-desktop`

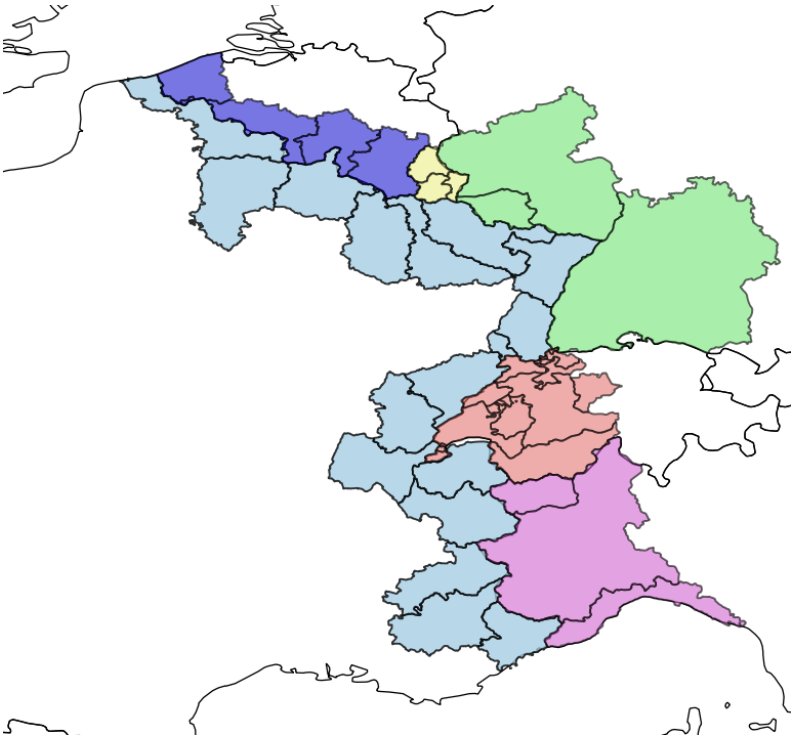


Figure 14: Overview of two datasets used for the Experiment #1. The blue dataset is a subset of GEOFLA, and the neighbouring countries (from top to bottom: Belgium, Luxembourg, Germany, Switzerland and Italy) are a subset of VMAP0.

large snapping value was used because of the large gap near Geneva). Both our prototype and FME took about the same processing time (0.5s versus 0.6s).

We have been able to align the boundaries successfully with our implementation, that is all output polygons are valid and there are no gaps nor overlaps along the boundary of France and its neighbours. With FME, there are several cases where there are gaps and overlaps, and several parts of polygons have collapsed, creating invalid geometries. We show in the Figure 15 the results for some specific locations. First, observe that in Figure 15b there are still gaps and overlaps. These are mostly caused because only slave vertices are snapped to the master primitives without the insertion of extra vertices. With our implementation, the gaps are split into different regions and the alignment is performed without leaving any problematic regions. Second, observe in Figure 15e how snapping makes the slave move to the wrong boundary (that of an administrative unit that is not on the boundary of the dataset), and how the very large gap is only partly aligned (some parts of it are within the given tolerance, others not). A large tolerance would have most likely solved this issue, but would also have modified the boundaries of polygons unacceptably more in other parts of the dataset. Third, it can be seen in Figure 15h that part of the boundary between 2 French polygons is collapsed to the boundary of the master dataset; in our implementation this boundary is left intact. Fourth, in Figure 15k, an enclave is snapped to the master dataset, which actually separates the polygon containing the enclave into unconnected parts.

4.2 Experiment #2: Conflation with an international boundary

As shown in Figure 16, we have extracted the international boundary from the VMAP0 and used it as input. The results of the aligned GEOFLA is exactly the same as in the previous section.

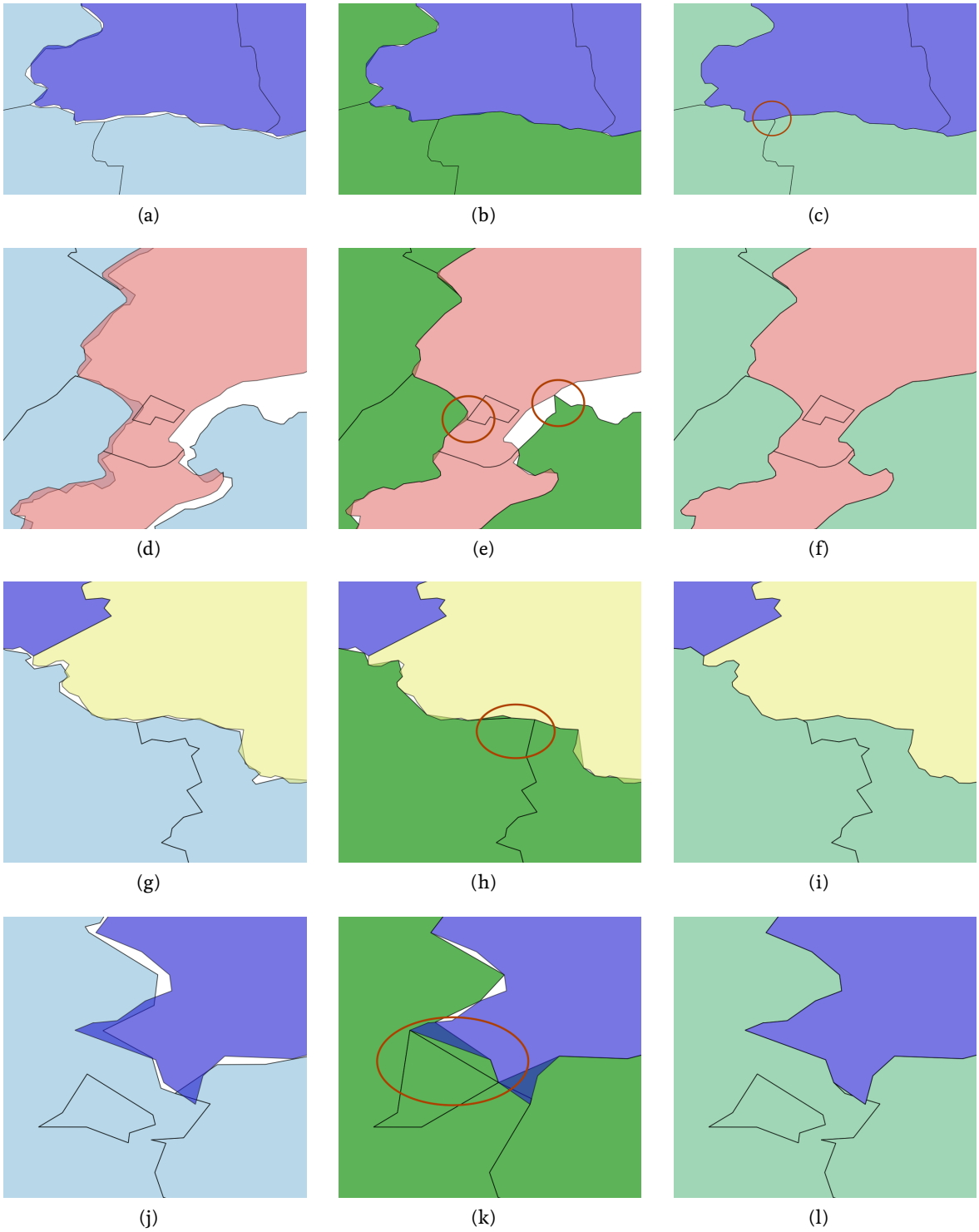


Figure 15: Left column: input (parts of Figure 14). Middle column: results with snapping. Right column: results with our implementation.

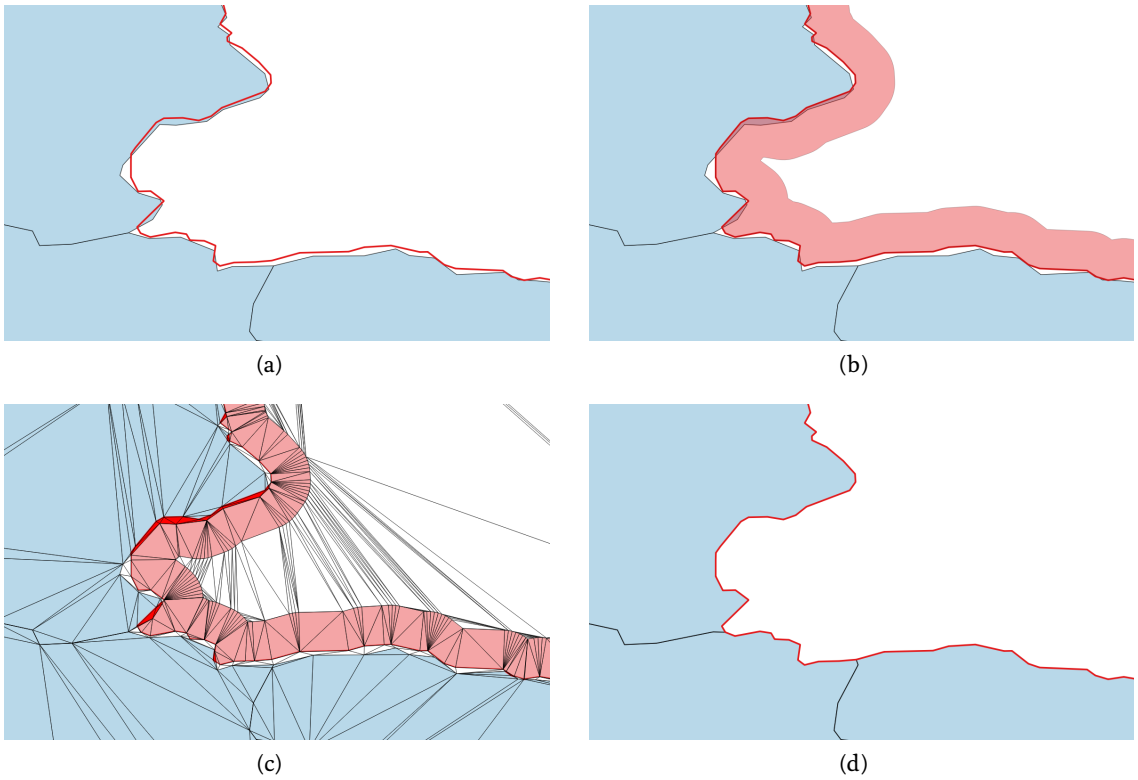


Figure 16: **(a)** Part of the GEOFLA dataset with an international boundary (red line). **(b)** Creation of a polygon from the boundary (red polygon). **(c)** The labelled CDT. **(d)** GEOFLA correctly aligned to the boundary.

4.3 Experiment #3: Spatial extent

We have performed a hierarchical conflation with the hierarchy of a dataset from the *Statistics Netherlands*[9]. The dataset, let us call it CBS2009, is shown in Figure 17 and is publicly available[10] The 441 municipalities are divided into 2,542 districts, and these are further divided into 11,574 neighbourhoods (many of these objects are `MultiPolygons`, as explained in Section 3.4). Each polygon has an identifier linking it to its parent. The conflation was performed in 2 steps: (1) all the district polygons with a given identifier are aligned to their respective municipality polygon (thus 441 times); (2) all the neighbourhood polygons with a given identifier are aligned to their respective (aligned and modified) district polygon (thus 2,542 times). As an indication, the whole process—2,983 boundary alignments—took 9m50s (4m45s computation; 5m05s for reading/writing to disk the datasets). While there were no very large errors, 98/441 municipalities and 2,213/2,542 districts were not properly subdivided (often by only a millimetre or less).

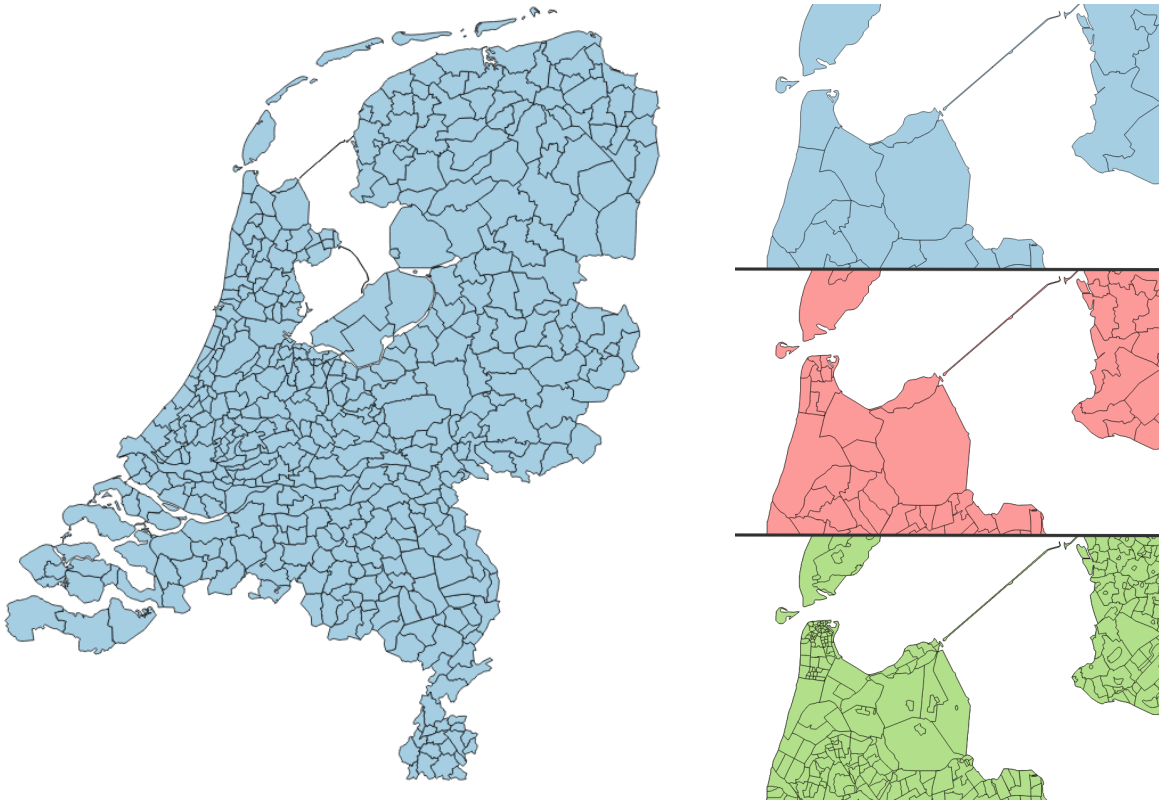
5 Conclusions

We have proposed a new algorithm to perform the horizontal conflation and we have shown that it is suitable for several cases that practitioners face in their work. It can be used with polygons and lines, and supports the alignment of polygons to a spatial extent. Furthermore, the conflation can be performed with a *local* criteria, instead of a global one (the user-tolerance is usually for the the whole dataset).

Our approach 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

[9] CBS (*Centraal Bureau voor de Statistiek*)

[10] `http://www.cbs.nl/nl-NL/menu/themas/dossiers/nederland-regionaal/publicaties/geografische-data/archief/2010/2010-wijk-en-buurtkaart-2009.htm`



(a) (b)

Figure 17: **(a)** Municipalities of the Netherlands (CBS2009 dataset). **(b)** Each municipality is subdivided into districts and then into neighbourhoods.

constructed, which allows practitioners to use the output for further analysis. Anyone who has tried—and perhaps failed—to find the appropriate snapping threshold for a given dataset by using trial and error will recognise that our approach has great benefits. One can get an overview of which polygons were modified (including what part(s) of these, and their exact geometry), and collapsed geometries are completely avoided (which happen often with large snapping tolerance).

We plan in the future to modify the algorithm so that the conflation of only lines is possible, eg for highways or rivers crossing a boundary. These would be "linked", with edges of the CT, although the use of a tolerance would still be necessary. We also plan to add more repair functions, particularly one where we can edge-match two polygons without the notion of a master and a slave and thus distribute errors. Triangles could be used to find the centreline of a region, as Bader and Weibel [1998] showed.

Acknowledgements

This research was supported by the European Location Framework project (`www.elfproject.eu`).

References

Arroyo Ohori K, Ledoux H, Meijers M (2012) Validation and automatic repair of planar partitions using a constrained triangulation. Photogrammetrie, Fernerkundung, Geoinformation 1(5):613–630

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

Beard KM, Chrisman NR (1988) Zipper: A localized approach to edgematching. Cartography and Geographic Information Science 15(2):163–172

de Berg M, van Kreveld M, Overmars M, Schwarzkopf O (2000) Computational geometry: Algorithms and applications, 2nd edn. Springer-Verlag, Berlin

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

Butenuth M, v Gösseln G, Tiedge M, Heipke C, Lipeck U, Sester M (2007) Integration of heterogeneous geospatial data in a federated database. ISPRS Journal of Photogrammetry and Remote Sensing 62(5):328–346

Chew L (1987) Constrained Delaunay triangulations. In: Proceedings of the third annual symposium on Computational geometry, pp 215–222

Davis M (2003) Java conflation suite. Tech. rep., Vivid Solutions, draft available at `http://www.vividsolutions.com/jcs/`, has never been completed.

Doytsher Y (2000) A rubber sheeting algorithm for non-rectangular maps. Computers & Geosciences 26(9–10):1001–1010

Gillman DW (1985) Triangulations for rubber-sheeting. In: Proceedings Autocarto 7, pp 191–199

Hastings JT (2008) Automated conflation of digital gazetteer data. International Journal of Geographical Information Science 22(10):1109–1127

INSPIRE (2008) Drafting Team "Data Specifications"—deliverable d2.6: Methodology for the development of data specifications. Tech. Rep. Identifier: D2.6_v3.0, Infrastructure for Spatial Information in Europe

INSPIRE (2014) D2.8.i.4 data specification on *Administrative Units*—technical guidelines. Tech. Rep. Identifier: D2.8.I.4_v3.1, Infrastructure for Spatial Information in Europe

Klajnšek G, Žalik B (2005) Merging polygons with uncertain boundaries. Computers & Geosciences 31(3):353–359

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

OGC (2006) OpenGIS implementation specification for geographic information—simple feature access. Open Geospatial Consortium inc., document 06-103r3

Ruiz JJ, Ariza FJ, Ureña MA, Blázquez EB (2011) Digital map conflation: a review of the process and a proposal for classification. International Journal of Geographical Information Science 25(9):1439–1466

Saalfeld A (1988) Conflation—automated map compilation. International Journal of Geographical Information Systems 2(3):217–228

Schuurman N, Grund D, Hayes M, Dragicevic S (2006) Spatial/temporal mismatch: a conflation protocol for canada census spatial files. The Canadian Geographer 50(1):74–84

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

Siejka M, Ślusarski M, Zygmunt M (2013) Verification technology for topological errors in official databases with case study in Poland. Survey Review 46(334):50–57

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

Zygmunt M, Siejka M, Ślusarski M, Siejka Z, Piech I, Bacior S (2014) Database inconsistency errors correction, on example of LPIS databases in Poland. Survey Review 47(343):256–264