INTERNATIONAL JOURNAL OF GEOGRAPHICAL INFORMATION SCIENCE, 2016
VOL. 30, NO. 8, 1522–1551
http://dx.doi.org/10.1080/13658816.2016.1140770

SplitArea: an algorithm for weighted splitting of faces in the context of a planar partition

Martijn Meijers[a], Sandro Savino[b] and Peter Van Oosterom[a]

[a]TU Delft - Faculty of Architecture, OTB, section GIS Technology, Delft, Netherlands; [b]University of Padova - GIRTS Lab, Padova, Italy

ABSTRACT
Geographic data themes modelled as planar partitions are found in many GIS applications (e.g. topographic data, land cover, zoning plans, etc.). When generalizing this kind of 2D map, this specific nature has to be respected and generalization operations should be carefully designed. This paper presents a design and implementation of an algorithm to perform a split operation of faces (polygonal areas).

The result of the split operation has to fit in with the topological data structure supporting variable-scale data. The algorithm, termed SplitArea, obtains the skeleton of a face using a constrained Delaunay triangulation. The new split operator is especially relevant in urban areas with many infrastructural objects such as roads. The contribution of this work is twofold: (1) the quality of the split operation is formally assessed by comparing the results on actual test data sets with a goal/metric we defined beforehand for the 'balanced' split and (2) the algorithm allows a weighted split, where different neighbours have different weights due to different compatibility. With the weighted split, the special case of unmovable boundaries is also explicitly addressed.

The developed split algorithm can also be used outside the generalization context in other settings. For example, to make two cross-border data sets fit, the algorithm could be applied to allow splitting of slivers.

ARTICLE HISTORY
Received 9 October 2014
Accepted 20 December 2015

KEYWORDS
Generalization; vector data modellingscale; topological data model; triangulated irregular networks

1. Introduction

A vario-scale structure is an integrated representation of geographic information over a range of scales, having the key property that a small change in the scale (dimension) leads to a small change in representation of geographic features that are represented on the map. From this structure, continuous generalizations of real-world features can be derived, which can be used for presenting a smooth zoom action to the user. To obtain a vario-scale map, an area-partitioning hierarchy is created starting from the most detailed topological base representation and applying generalization operators such as simplification, selection and elimination (Van Oosterom 2005). The result is stored in a tree structure of polygons (generalized area partitioning (GAP)-tree) and can be used afterwards efficiently to select a representation at a specific map scale. To enhance the

CONTACT Martijn Meijers b.m.meijers@tudelft.nl

cartographic quality of the maps that can be obtained from the tree structure, the ‘split operator’ was proposed (Ai and Van Oosterom 2002). Instead of assigning the area of the least important object completely to one of the neighbours, resulting in a binary tree structure, the area is more or less fairly distributed among the neighbours based on compatibility. The resulting hierarchical structure is no longer a binary tree, but a directed acyclic graph (DAG) as one object is split over multiple neighbours (and has multiple parents in the structure). However, this potentially led to the problem of important feature(s) such as roads being split and lost. Further, only limited qualitative and quantitative assessments were presented.

In context of the GAP-tree generalization, the problem of well-represented infrastructure objects, such as roads, rivers, canals and railroads (often represented as elongated areas), at the smaller scale ranges of the vario-scale structure was not well solved (Van Putten and Van Oosterom 1998). The purpose of the presented research (algorithm) is to produce better-quality vario-scale maps. One of the identified main problems with the vario-scale approach is the handling of linear infrastructure features, such as roads, waterways, etc. The proposed algorithm can be used to compute this linear representation and at the same time split the old area and assign this in a ‘fair’ manner to the neighbouring area features. The requirement that the result should fit in the topological data structure is inherent in the design of the algorithm (see Figure 1).

However, in certain weighted split situations, some neighbours should take no share of the subdivided feature due to extremely incompatibility. Besides the design and implementation of the algorithm, a measure of a balanced weighted split is proposed. The algorithm is demonstrated to function in all conceivable situations. The quality of the results of our algorithm is evaluated based on the criterion of the balanced weighted split, that is, how close does our algorithm approach the balanced split, whereby each neighbour gets a share of the splittee exactly proportional to its weight. Further, the effect of the new weighted split algorithm is assessed by comparing the obtained results with similar results obtained by the approach for creating a vario-scale map. In the research described in this paper, the algorithm is applied to all features (as ultimate stress test). In future work, this could be limited to a specific type of features (e.g. based on classification) or features with specific shape (e.g. by looking at area vs perimeter). For other features, the standard merge operator can be used.

The remainder of the paper is structured as follows. After an overview of related work in Section 2, the algorithm and the metric used to assess the results are explained in Section 3. Section 4 explains the weighted split. Section 5 shows and discusses the

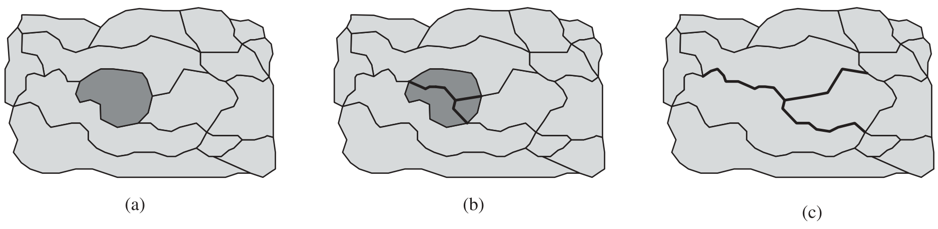


Figure 1. By applying a split operator, a polygon can be fairly distributed among its neighbours. (a) Polygon to be split. (b) New boundaries. (c) Result of split operation.

results we obtained with SplitArea. Section 6 concludes the paper and gives some suggestions for future work.

2. Related work

In the past, different methods have been proposed for reducing the dimensionality of polygonal features in maps, thus 'collapsing' the polygonal representation to one that is based on lines and obtaining the skeleton[1] of a shape.

Probably the best known method in this respect is the medial axis transform, first described by Blum (1967). Approximations of this skeleton are often computed for shape recognition in a raster environment, for example, Attali *et al*. (1995) and Bai and Latecki (2007). Another example, more related to maps and vector representations, is found in Jones *et al*. (1995), where a triangulation is used to perform map generalization. Their collapse operation is 'used to reduce the dimensionality of an areal object either to a line or to a point'. Information is only provided on how to obtain a thinned representation with the triangulation, but not how to adapt the neighbouring objects to this new representation (nor is there any discussion of the splitting of objects with different weights for attractiveness).

Uitermark *et al*. (1999) used a similar application of a triangulation. Their method describes the use of a constrained Delaunay triangulation (CDT) to obtain the centre lines of a road network, where the road area is described with polygons. Triangles in the CDT can be classified by determining the number of constrained edges in the boundary of a triangle. Based on this classification, the centre lines of the road network can be found. Despite being a little different from our problem (splitting of objects), the classification of the triangles used is applicable.

Also, Regnauld and Mackaness (2006) use this classification of triangles to calculate the centre lines of a river network. For more details, see Section 3.3.3. Although not explicitly stated, it is also this way of obtaining the skeleton that is used by Bader and Weibel (1997) to split a polygon between its neighbours. Bader and Weibel mention that the skeleton obtained is not necessarily connected to the rest of the line work describing the boundaries. The approach described certainly looks promising but did not address how to deal with different weights, or how to block a neighbour from getting a share of the area to be split.

McAllister and Snoeyink (2000) take another approach for obtaining the centre lines of a river network. The network in their approach is represented as polygonal lines that are tagged as either left or right banks (describing a polygonal area). They use a robust implementation of the Voronoi diagram and a topological data structure (the quad-edge data structure) to obtain an approximation of the medial axis of the river network to perform analysis. They observe that their implementation cannot deal with cycles (holes, which are islands in the river) in the obtained approximation, because that was not needed in their application. We want to have a more generic solution which is also able to deal with area features with holes. In addition, weights are not considered as the centre line is what they want to calculate.

Gold and Snoeyink (2001) describe another method based on the Voronoi diagram (dual to the Delaunay triangulation) that can at the same time find the crust (boundary) and the skeleton of a polygonal shape only described by a sufficiently dense sample of points. Although we could add additional vertices to the input, the skeleton obtained might contain a lot of branches and will not be connected to the boundary of the neighbouring polygons of the polygon to be split. It is also unclear how a weighted skeleton should be obtained by this method.

Ai and Van Oosterom (2002) discuss the use of skeletonization for splitting in the context of a hierarchical structure (the GAP-tree, which was not based on topological primitives but on geometry). Although they mention the usefulness of weights for splitting polygon objects over different neighbours with different classifications, no special cases are mentioned for dealing with neighbours which are incompatible and should not get a share at all. Also, only some cartographic examples of the results were given, but no formal assessment of the quality of these results took place. Furthermore, their implementation is not described in detail.

Haunert and Sester (2008) have used the straight skeleton for obtaining centre lines for a road network. They also describe the possibility of using this type of skeleton as a (partial) collapse operator. Although their work mentions that it is possible to create a weighted version of the straight skeleton, they do not mention unmovable boundaries, which are important in our setting.

Similarly, Szombara (2013) proposes a method using the medial axis transform that indicates which parts of an area should be collapsed and which parts of a shape should be enlarged to be visible on a smaller scale map. However, as the medial axis transform (obtained by computing the Voronoi diagram of the boundary segments) is used, the skeletal representation obtained lies exactly in the centre of the area.

3. The SplitArea algorithm: an explanation

This section gives an overview of the SplitArea algorithm. The purpose of the algorithm is to split a face of a planar partition among its neighbouring faces. We first discuss the requirements for the algorithm, followed by a discussion on how to judge whether a split is performed in a fair way. Then we discuss the steps of the algorithm in detail and show that we can also handle holes in the face to be split.

3.1. *Requirements*

Figure 2 illustrates the set of requirements we formulated that should be present in such a splitting operation, taking into account the topological data structure:

(1) Assign larger pieces of the area object to be split to more compatible neighbours and smaller pieces to less compatible ones.
(2) Prevent certain merges, that is, completely disallow merges between faces having extreme incompatible feature classes. A special variant of this requirement is that the extent of our geographic domain covered by the data set should not be changed, which means that the objects should not be merged to the universe (everything outside the domain covered by the planar partition of the data set).
(3) The resulting line work should fit in and connect to the rest of the planar partition.
(4) Deal with holes (filled by one or more island faces that can also get a share of the face to be split).

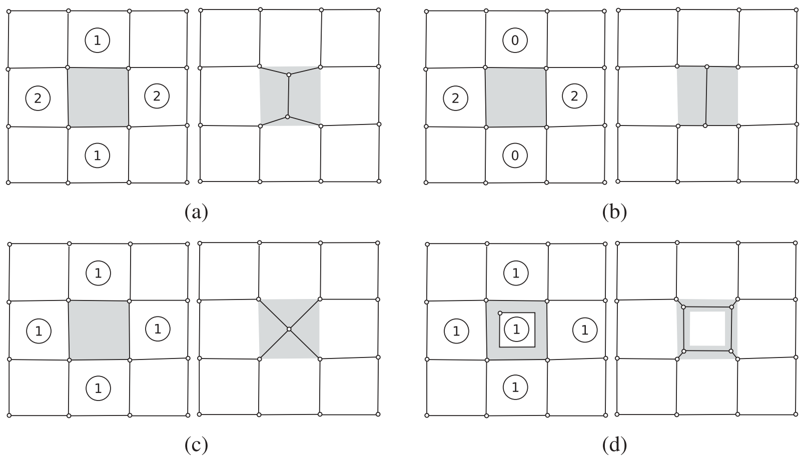


Figure 2. Requirements for our split operation (area object to be split is shown in grey). Circled numbers are the attractiveness values of the neighbours. (a) Larger pieces should be assigned to more attractive neighbours. (b) Certain neighbours should be prevented from getting a share. (c) Resulting line work should fit in and connect to the remaining boundaries. (d) Input features might have holes and splitting should take these holes (containing another feature) in consideration.

(5) The method should work on vector data stored in a topological data structure (such as the well-known node–arc–face data structure, cf. Worboys and Duckham (2004)) and generate the topological references (e.g. incidence relations between edges and faces) for the edges being output.

3.2. *Judging the balance of a split*

Intuitively, semantic similarity of features should determine the outcome of generalizing area objects. A lot of research has been devoted to determine when objects are compatible so they can be aggregated (e.g. Van Smaalen 2003, Vermeij *et al.* 2003, Yaolin *et al.* 2003, Hofman 2008, Schwering 2008, Haunert and Wolff 2010, Yang *et al.* 2013). In the original GAP-tree (Van Oosterom 1993), the compatibility between a specific face and one of its neighbours, neighbour$_i$ (at instance level), is defined with three terms to determine with which neighbour a face should be merged:

(1) Type compatibility ('compatibility' in Equation 1). This takes into account the semantic distance between the feature class of the neighbour and the splittee. For example, urban fabric can be considered more compatible with industrial area than with grassland. Objects having similar feature classes are thus considered more compatible. This is encoded by a value between 0 and 1, where 0 means not compatible at all and 1 means totally compatible. The compatibility values can be stored in a square matrix for all feature classes, which is termed the compatibility matrix.
(2) Boundary length between the splittee and neighbour$_i$ ('length' in Equation 1).
(3) The relative importance (or weight) of the feature class of neighbour$_i$ ('imp' in Equation 1). Dependent on the use of the data set, different feature classes can be

given a higher weight, which means they are less considered for being generalized, compared to equal-sized features having a lower weight.

Now that we want to *split* faces, we can use the same terms for defining how we want to fairly split a face over its neighbours. Equation 1 contains these terms for an attraction value for neighbour$_i$ of the splittee *s*:

$$\text{attraction}(\text{ngb}_i) = \text{compatibility}(\text{s}, \text{ngb}_i) \times \text{length}(\text{s}, \text{ngb}_i) \times \text{imp}(\text{ngb}_i) \tag{1}$$

The more attractive a neighbour is, the larger share it should receive from the split area. The attractiveness measure tries to influence the position of the newly generated skeleton segments. This way, the attractiveness values are used at processing time to make sure the split is performed as balanced as possible. Figure 2(a) illustrates that the neighbours that have an attraction value of 2 get a bigger share than the neighbours that have an attraction value of 1.

After splitting the face, the initial attraction values can also be used to evaluate the result of the split algorithm. Equation 2 shows that we expect a neighbour *i* to get a share of the splittee *s*, which is in accordance with the sum of all attraction values of the *n* neighbours around the splittee. In case of Figure 2(a), the left and right neighbour should get $\frac{2}{6} = \frac{1}{3}$ of the splittee, while the top and bottom neighbour should get $\frac{1}{6}$ each.

$$\text{share}(\text{ngb}_i)_{balanced\_target} = \frac{\text{attraction}(\text{ngb}_i)}{\sum_{j=0}^{n} \text{attraction}(\text{ngb}_j)} \times \text{area}(\text{s}) \tag{2}$$

The share that a neighbour *i* actually obtains is easily measured by subtracting the size of the neighbour before the split from the size of the neighbour after the split (Equation 3).

$$\text{share}(\text{ngb}_i)_{obtained} = \text{area}(\text{ngb}_i)_{post} - \text{area}(\text{ngb}_i)_{pre} \tag{3}$$

We now can obtain the absolute difference of the share that one neighbour should obtain in theory and the share that it gets assigned by SplitArea (Equation 4). This difference is the error that is made in appointing a share to one of the neighbours.

$$\text{error}_i = abs\left(\text{share}(\text{ngb}_i)_{obtained} - \text{share}(\text{ngb}_i)_{balanced\_target}\right) \tag{4}$$

Equation 5 shows that summing all error values of the neighbours involved in the split operation, normalized against the size of the splittee *s*, leads to a value that expresses the total error made with subdividing the feature.

$$\frac{\frac{1}{2} \times \sum_{j=0}^{n} \text{error}_j}{\text{area}(\text{s})} \tag{5}$$

Because for each error the value is recorded twice (what one neighbour gets too much is automatically not only appointed to any of the others but also counted), the total error value is divided by two, resulting in an error value between 0 and 100%.

Although the attraction values are an attempt to more formally express what a balanced split is, we do not take into account some aspects of the resulting geometry (e.g. roughness of remaining line work or number of vertices in the newly created boundaries). Such 'quality indicators' are out of scope for the work performed here but could be considered later in a similar manner.

3.3. *Explanation of the algorithm*

The algorithm uses a CDT to find a skeleton inside the face to split, prunes parts of this skeleton and connects the obtained line work to the rest of the planar partition. The following definitions will be used (illustrated in Figure 3):

Face: A polygonal object, possibly with holes, representing one object of the planar partition. In the topological structure, the geometry of a face is not explicitly represented as a polygon, but its geometry has to be obtained from the set of edges (straight-line boundaries) related to it.

Topological chain: An edge of the planar partition. Every topological chain lies exactly between a left and a right face. Topological chains in the structure are modelled as

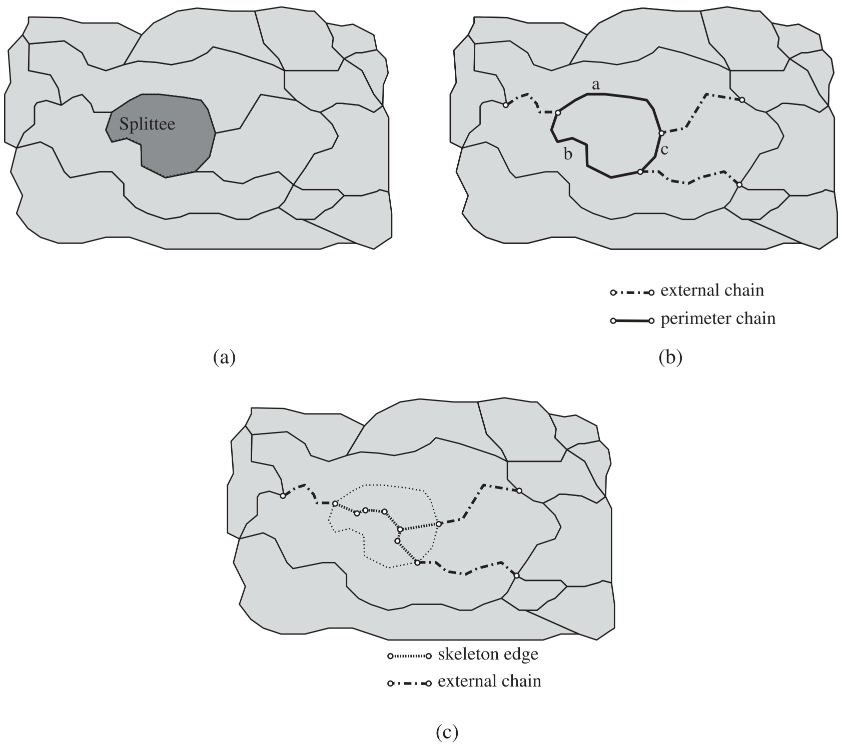


Figure 3. An example showing SPLITAREA at work. (a) The splittee: the object that has to be subdivided over its neighbours. (b) The perimeter chains (labelled a, b, c) and the three external chains (chains incident with the perimeter chains). (c) Inside the splittee, skeleton edges will be created. The new boundaries will be formed by these edges and the external chains.

polylines, having a start node and an end node reference, as well as two face references (one for the left neighbouring face and one for the right).

Splittee: The face *F* for which the area has to be split over the neighbouring faces, cf. Figure 3(a).

Perimeter chain: A topological chain having the face *F* as its left or right face.

External chain: A topological chain that touches at least one perimeter chain of the face *F*. An external chain is *not* part of the boundary of the splittee *F*.

Skeleton edge: The skeleton is the collection of line work that is created inside the splittee. The skeleton consists of a set of edges (line segments having two points). The new boundaries for the neighbours of the splittee will be formed by the set of skeleton edges and the connections to the external chains (see Figure 3(c)).

The input of the SplitArea algorithm is:

- The id of the face *F* – the splittee
- The set perimeter chains of the face *F*
- The set of external chains of the boundary of face *F*
- In case of weighted split, compatibility and class weights (Equation 1)

The output of the algorithm consists of a set of new topological chains representing the new boundaries of the splittee's neighbouring faces: some of these topological chains can be completely new, others are an extended version of the external chains. Both the input and output topological chains contain their left and right references; as we will see in Section 4, the input chains may have also a weight value attached to them.

The algorithm performs the following steps (Sections 3.3.1–3.3.6):

(1) Triangulation
(2) Selection of internal triangles
(3) Creation of the skeleton
(4) Creating connectors
(5) Edge labelling and skeleton pruning
(6) Obtaining the final boundaries

After these steps are described, we give an overview of how to handle holes in the splittee in Section 3.4. Our algorithm solves a new problem (weighted split and unmovable boundaries within a planar partition) of which the skeleton built by Delaunay triangulation is a special case (equal weights and no unmovable boundaries).

3.3.1. *Triangulation*

The first step of the algorithm is to build a CDT of the perimeter chains of the splittee.

A constrained triangulation is a triangulation of a set of points that has to include a given set of segments between these points. As constrained edges are not necessarily Delaunay edges, a CDT tries to fulfil the empty circumcircle property as much as possible, but for a triangle with a constrained edge, the empty circle criterion is weakened (see Shewchuk 2008 and Shewchuk and Brown 2015 for more details on how to construct a CDT). Note that the set of constrained triangulation edges is the boundary of the splittee.

3.3.2. *Selection of internal triangles*

The triangulator produces a mesh of triangles both in the interior and exterior of the splittee, bounded by a triangle with points at infinity (Liu and Snoeyink 2008). To select only the interior triangles, the algorithm performs a walk on the triangles (see Figure 4).

The walk starts from one of the triangles having one vertex at infinity. While walking from one triangle to an adjacent one, the triangulation edge that the two triangles have in common is crossed: if this edge is a constrained edge, then we know that we are on the interior of the splittee. Once inside, we can flag the triangles we are looking for as internal – by avoiding crossing a constrained edge again, the walk stays inside.

3.3.3. *Creation of the skeleton*

The creation of the skeleton edges is performed individually for each triangle, following the technique described in Uitermark *et al.* (1999). Uitermark et al. also use a CDT algorithm in which they input a graph $G = (V, E)$, where V is a set of vertices, the end points of the input edges, and E is this set of edges which they termed *G-edges* (constrained triangulation edges). Looking at the resulting CDT, two kinds of edges can be distinguished: G-edges, edges that were already present in the input graph, and D-edges, created by the CDT algorithm. Triangles in the CDT can now be classified by determining the number of G-edges in the boundary of a triangle. In this way, four types of triangles can be distinguished: 0-triangles, 1-triangles, 2-triangles and 3-triangles, where the number is the number of G-edges (i.e. edges also present in the input). A skeleton of the triangulated object can be found by forming new line segments based on connecting the geometric midpoint of the *D-edges* for the triangles (see Figures 5 and 6(a) for an illustration).

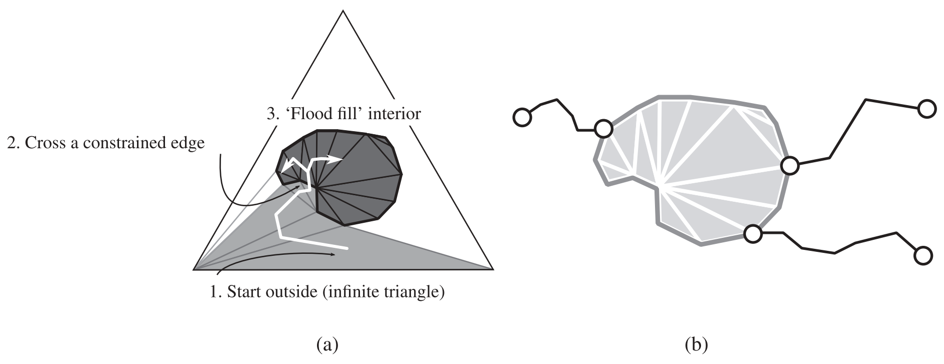

3. 'Flood fill' interior
2. Cross a constrained edge
1. Start outside (infinite triangle)
(a)
(b)

Figure 4. Selection of internal triangles. (a) Walk on internal triangles. (b) Triangulation of the splittee, where all internal triangles have been selected. Also external chains are shown.

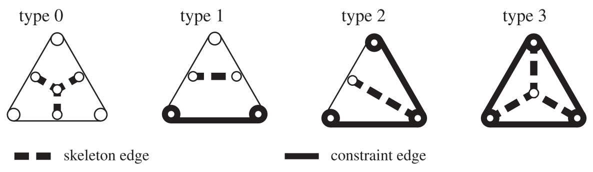


Figure 5. Four types of triangles can be distinguished by looking at the number of G-edges: 0-triangles, 1-triangles, 2-triangles and 3-triangles. By connecting the midpoints of the D-edges (unconstrained edges), a skeleton edge can be obtained (skeleton edges are visualized with dashed lines).

3.3.4. *Creating connectors*

To complete the construction of the skeleton, we need to assure that it is connected with the existing line work outside the boundary of the splittee, that is, it has to be connected with the external chains, see Figure 6(b).

Every external chain touches the splittee in one or two nodes: if such a node is not already connected to the skeleton, then a special skeleton edge is drawn from this node to the rest of the skeleton. These special edges are called connectors and are generated according to the scheme shown in Figure 7(a). Figure 7(b) shows that by generating the connectors only going outward of a node when going counterclockwise around a triangle, we can guarantee that this operation is local (so no need to look at neighbouring triangles when generating skeleton edges and connectors) and that duplicate connectors in neighbouring triangles are prevented. Another advantage of creating connectors this way is that no intersections between connectors will be generated (similar to Kieler *et al.* 2009).

Depending on the triangulation, there can be many unconstrained triangulation edges between the skeleton and the node of an incident external chain: in such a case, the connector to be used has to be chosen. In our implementation, we decided to choose the connector for which the angle is the most collinear with the direction of the external chain, but other choices could be preferred (e.g. the longest or the shortest one).

3.3.5. *Edge labelling and skeleton pruning*

The next step of the algorithm is to propagate the left and right face reference values from the external chains to the skeleton edges. For this purpose, a graph is built in

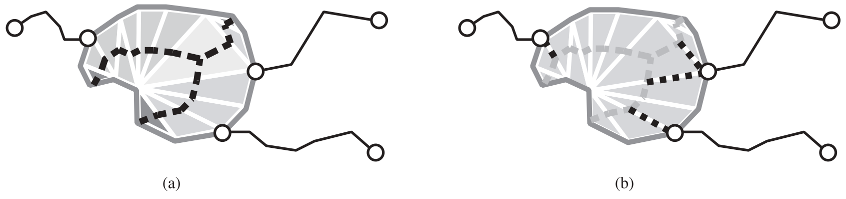
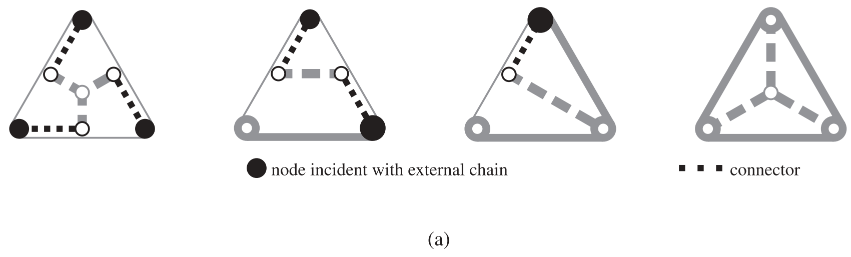


Figure 6. Creation of the skeleton. (a) Obtaining skeleton edges using the triangulation. Note: triangle type is indicated by grey tone (cf. Figure 7(b)). (b) Adding extra skeleton edges (connectors) to guarantee a connection between the existing external chains (those need to be preserved) and the skeleton. Note that a connector has to be chosen for the node touching the external chain at the top right.



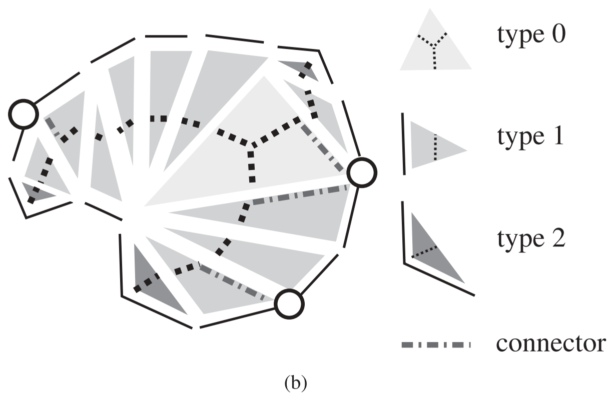


Figure 7. Connectors are created if a corner vertex of a triangle also is a node in the planar partition (skeleton edges are visualized with dashed lines). (a) The four different triangle types and how connectors are generated per type. (b) Skeleton edges and connectors are created locally (depending on triangle type) and duplicate connectors for neighbouring triangles are avoided.

which all skeleton edges, chosen connectors and external chains are represented. This graph can be represented by a winged-edge structure (Baumgart 1975).

The winged-edge structure is one of the most well-known data structures for storing topology and is characterized by having references at the begin node and the end node of each edge to first connected edge on both left and right sides. These edge-to-edge references can be deduced from the angle that segments have around a node and are very useful for efficient traversal and modification of the structure. The labelling starts at one side of an external chain, propagating the correct neighbour reference value onto all skeleton edges, until another external chain is encountered (for correct traversal, the edge-to-edge references are used). Then, the neighbour reference is switched, propagating the other reference from the external chain, and the labelling continues, until the external chain where the labelling started is encountered. An illustration of this labelling procedure is found in Figure 8.

At the end of the labelling, some of the skeleton edges can have the same neighbour reference on both their sides. This means that in the planar partition, these parts will be completely enclosed by one neighbouring face, therefore these edges are removed. Removal of these edges can be seen as pruning branches in the graph. The winged-edge structure makes it easy to delete those edges from the graph, while keeping the connectivity between the remaining edges in the graph. The removal of the branches leads to some artefacts (spikes, where the skeleton makes a sharp turn). To compensate for that, some shortcut edges are introduced, replacing the two skeleton edges that

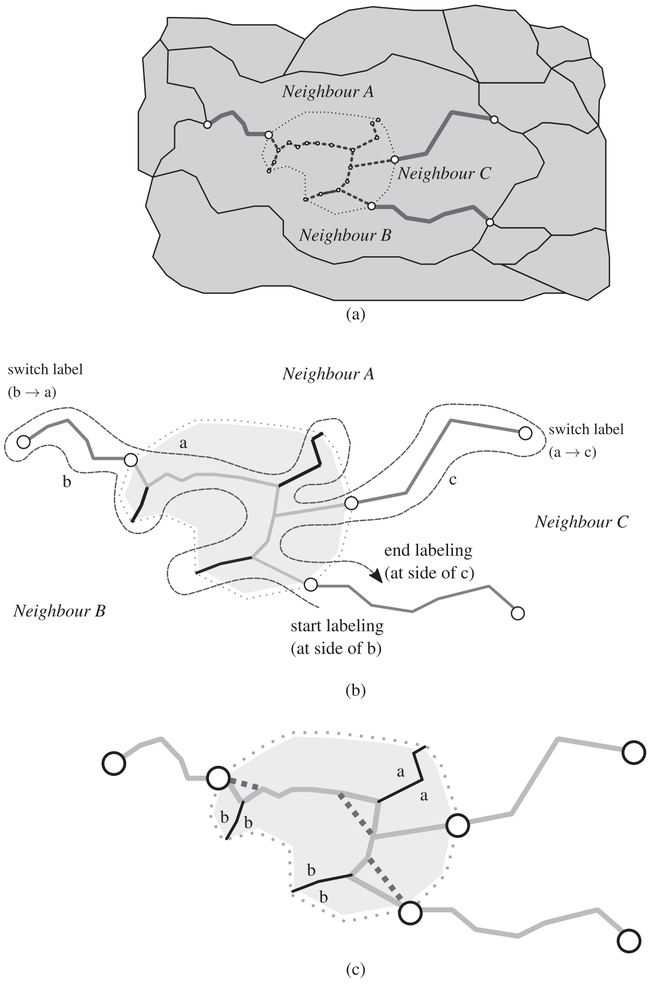


Figure 8. Labelling the skeleton edges with the correct neighbour reference and pruning parts of the skeleton that are enclosed completely by one neighbour. (a) All skeleton edges created (dashed, inside the splittee) and the external chains (dark grey). (b) Labelling of the created skeleton edges (light grey), starting from an external chain (dark grey). Note: edges that have the same neighbouring reference value on both sides once labelled are drawn in black. (c) Parts of the graph that have the same label on both sides (black) are removed. Shortcuts (dashed) replace the two skeleton edges that touch the branches that are removed.

make the sharp turn and are incident with the branch removed (see Figure 8(c)). The edges that are removed only belong to type-0 triangles, thereafter the shortcut edges that replace them do not affect the topology, as they are inside triangles that already belong to the original area.

3.3.6. *Obtaining the final boundaries*

As a final step, the new topological chains for use in the area partitioning are obtained. All skeleton edges and external chains that have the same neighbour reference values are merged into one topological chain. When this process has finished, all the perimeter edges

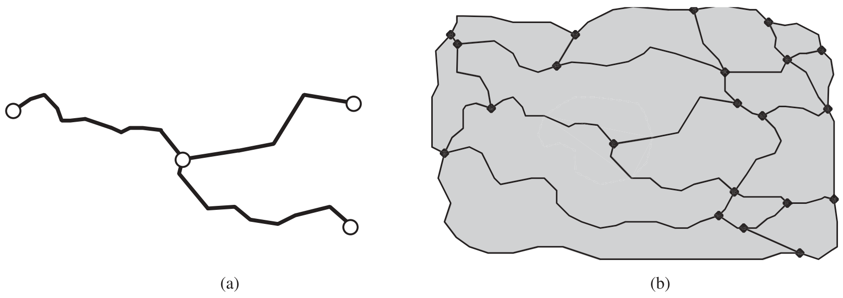


Figure 9. Result obtained with SplitArea (a) All skeleton edges and external chains are merged into the longest topological chains possible (having the same neighbour references). (b) The line work obtained fits in with the rest of the planar partition.

of the splittee can be removed from the structure, the new boundaries have been completely built and will fit in with the remaining part of the line work of the planar partition, and the area of the splittee *F* has been subdivided over its neighbouring faces (cf. Figure 9).

3.4. *Handling holes in the splittee*

One of our requirements was that the algorithm also should handle holes (i.e. island faces within the outer boundary of the splittee). In an area partition, a hole in an area feature contains another feature. When the surrounding feature is being processed by the split operation, the hole feature should be kept and even enlarged to get a fair share of the surrounding area.

Irrespective of whether the input contains holes, all segment geometry for the skeleton is obtained correctly by our approach from the classification of triangles (as described in Section 3.3.3 and illustrated in Figure 7). However, the generation of topological references needs attention when the input face contains a specific type of islands in its interior. For such islands, we term them 'lonely' islands here; one face is completely contained in a hole of the splittee.

Figure 10 shows three of such lonely island faces: faces B, C and D. As shown, the resulting skeleton for the splittee, face A, will not be a tree structure but will result in a graph having cycles (one cycle per island). This has consequences for the labelling step described, as not all edges in this graph can be labelled on both sides – the inside of the cycles cannot be reached automatically. No external edges will be present at the interior side of the cycle of a lonely island. This is only the case for lonely islands but not for holes containing more than one island face. In the example, the cycles for faces E and F will be automatically labelled from the boundary edge between the two (as this edge will be inserted as an external edge in the skeleton graph).

To solve the labelling problem, one node from a ring around a lonely island will be attributed with the face identifier that lies inside the ring. Subsequently, generated skeleton segments incident at those nodes will be marked as being a seeding segment and as such will be treated as external segments from which the label propagation can take off. Note that only one face pointer will be set on both sides of these external segments so that at the end of the labelling step they will be removed (see face B in the example).

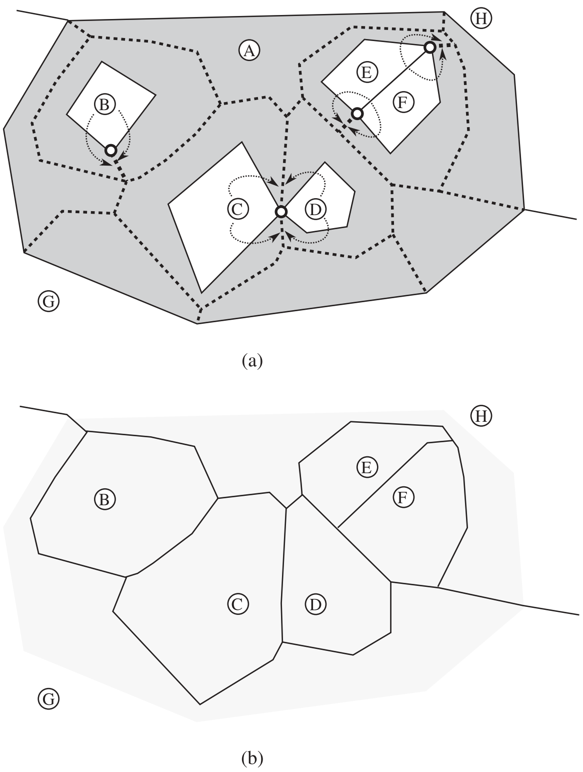


Figure 10. Handling islands when splitting a face (face A). No extra steps have to be taken for label propagation when a hole is filled by more than one face (cf. faces E and F). However, care has to be taken when holes are filled by only one face (face B), or when island faces are tangent in one node (faces C and D). (a) Careful labelling is necessary when handling islands. (b) Result.

Another typical case that happens in real-world data sets is that islands can be tangent at a point (e.g. bow-tie shape). In this case, this tangent point will be chosen as the node from which to generate the face pointers for the cycles. In this particular case, care has to be taken when propagating the adjacent face information to the incident seeding segments. This can be solved by taking into account how the original faces are incident to the node, thus looking at which face is incident to which sector around the node. Note that more than two faces can be tangent in one point. Based on the angle of the new segment and this sector information, we determine which face pointers have to be propagated onto the newly created segment. Two different face pointers will be set on the two sides of the external segment (see faces C and D in the example, one side of the segment will be labelled with C, the other side with D).

4. A weighted split

In Ai and Van Oosterom (2002), it was mentioned that setting weights could help to get a more balanced split, for example, objects having a similar feature class can get a larger share of the splittee than less compatible features. With the normal approach, the vertices of the skeleton edges to be created are normally positioned exactly at the middle of an unconstrained edge. This changes with the weighted approach.

4.1. *Introducing weights*

To obtain a weighted solution, weight values are given as input together with the perimeter chains. For each vertex that is part of the triangulation, at least one weight value will be present. The vertices of the skeleton edges can be slid along the unconstrained edges of the triangulation, respecting the weights. Figure 11 illustrates this. The weights make it possible to move the vertices of the skeleton edges further away from one perimeter chain (and closer to the opposite one).

The weights are determined by the compatibilities of the neighbours (see Equation 1). If the compatibility of a neighbour and the splittee is high, then the neighbouring face should get a large share of the face to be split. In this case, a higher weight is set on a perimeter chain compared to when the neighbour is less compatible with the splittee. A higher weight 'pushes' the newly created skeleton edge further away from the original perimeter chain, obtaining a larger share.

In the case that a vertex in the triangulation also is a topological node (an external chain touches the boundary of the splittee at this vertex), more than one weight will be associated with a vertex. Different choices can now be made on how to handle this multiplicity of weight values (we take the average, but also the minimum or maximum of the weight values could be chosen).

4.2. *Handling zero weights*

Putting weights on the perimeter chains in the input gives more flexibility on how to subdivide the area of the face to be split. To prevent one of the neighbours from getting a share at all, zero weights are introduced. This is useful in two cases: (1) prevent a neighbour that is highly incompatible with respect to the feature class of the splittee from getting a share and (2) when the feature to be split is at the border of the domain – we can then fix the border at its position (see Figure 12). Zero weights are in accordance with how weights are set up in the area-partitioning/topological structure. If two faces have two incompatible feature classes, a value of zero will be given in the compatibility matrix. Therefore, the resulting weight set on the boundary between two such faces will also be zero ($\text{ngb}_{compatibility} = 0$).

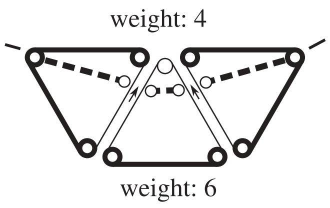


Figure 11. Weights are set on the perimeter chains and are entered in the triangulation at the vertices. The new vertices of the skeleton edges are moved along the unconstrained triangulation edge accordingly. The vertices where the external chains are incident have two weights set.

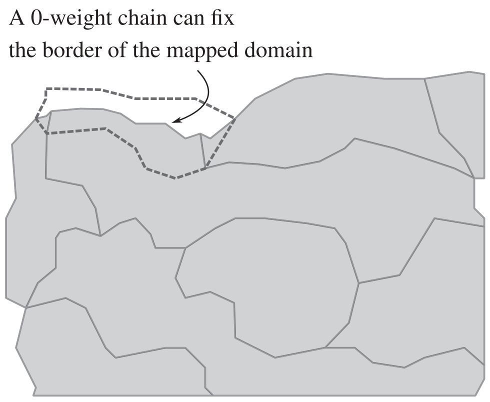


Figure 12. Fixing the border of the domain can be performed with zero weights.

When allowing zero weights as input, care has to be taken while generating the skeleton edges based on the classification of the internal triangles. The basic classification (0- to 3-triangle) has to be extended to deal with one, two or three vertices having a zero weight.

Figure 13 shows all possible situations that can occur with zero-weight vertices and Figure 14(b) shows some of these classified triangles in a real-world example. There are five cases that need some attention:

(1) Sometimes no skeleton edge has to be generated. This is the case with triangles with two vertices being on a constrained edge and the third vertex being a zero-weight vertex (Figure 13, triangle type-1-1-vertex-top, i.e. the topmost triangle of the three triangles in the cell belonging to triangle type-1 – row 2, column 1 in the figure – having one zero-weight vertex). See also Figure 14(b), triangle labelled with 't-1-1-v-top'.
(2) When two zero-weight vertices are opposite each other on an unconstrained triangulation edge, a skeleton vertex is generated at the middle of this edge, similar to when two weights are equal and non-zero (e.g. triangles type-1-3-vertex, type-2-2-vertex-bottom-left).
(3) As mentioned before, vertices in the triangulation have more than one weight associated if an external chain is incident (i.e. nodes in the planar partition topology). If one of these weights is zero, this zero weight has to be set as *the* weight of this vertex. This ensures that the vertices of the original perimeter chain are not moved.
(4) Introduction of zero-weight vertices can lead to cycles in the skeleton edge graph (see Figure 14(a) for an illustration). Therefore, in the labelling step, when left and right references are set, an extra check has to be performed. If after labelling, skeleton edges are encountered that are only labelled on one side and which were not entered in the triangulation as an edge with zero weight, these edges can act as a seeding edge from which the label can be propagated to the other side (the side of the edges that are interior to the cycle). This label propagation continues visiting all cycle edges until all these

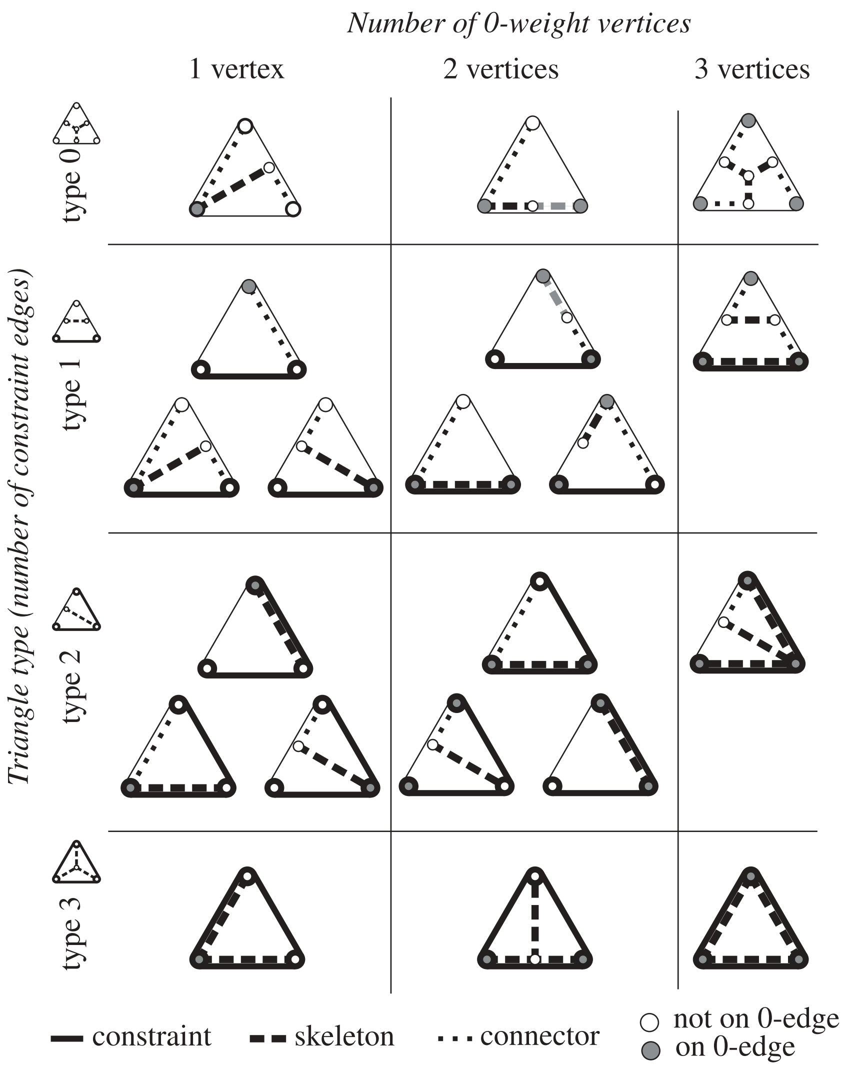


Figure 13. Triangles with zero-weight vertices. Unmovable vertices, having a weight = 0, are coloured grey. Skeleton segments that will be created are in black, skeleton segments that will not be created are visualized with grey (e.g. triangle type-0-2-vertices).

edges have been labelled on both sides. Note that edges that have the same label on both sides will be removed from the skeleton graph (just as in the unweighted case).

5. Results and discussion

To test the behaviour of SplitArea, we implemented[2] the algorithm in the context of the topological data structure using the Python programming language, while the data were stored in a PostgreSQL 9.2 database extended with PostGIS 2.0. The tests were run on our research server (an SUN V40z server running Solaris 10, 64-bit) equipped with 4 AMD

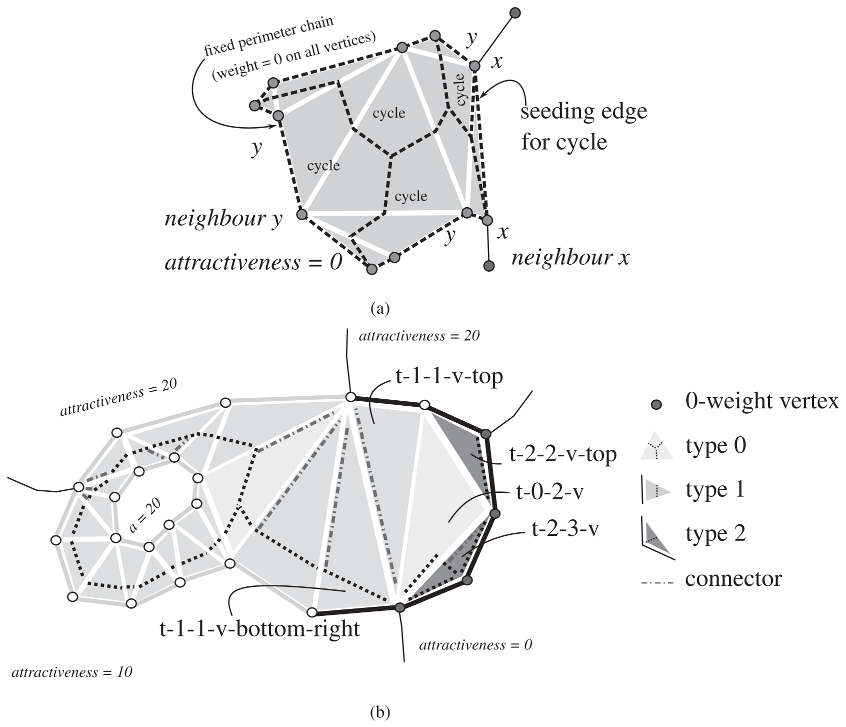


Figure 14. Fixed perimeter chains (i.e. zero-weight vertices) bring some special implications. This is illustrated with two cases. (a) Zero-weight edges can lead to cycles (corresponding to neighbour *y*, with fixed boundary) in the skeleton graph. The labelling step thus should be modified such that neighbour *x* gets the complete area of the splittee. (b) Also with zero-weight vertices, triangles are processed locally (independent of all other triangles). However, more possibilities have to be taken into account (cf. Figure 13). Note that triangles that have a zero-weight vertex have their constraints visualized in black and are labelled with their position in Figure 13.

CPUs clocked at 2.2 GHz and 8 GB of main memory. The algorithm was tested with two land-cover data sets. Table 1 shows some characteristics for both data sets and Figure 15 gives a graphical impression. As ultimate test, all polygons were split instead of merged to maximally test our algorithm with all kind of real-world data and a range of different polygon shapes. This is more an algorithm test and not an action to optimize cartographic quality of the generalized result (as then a choice between when it is best to merge and split should be made). It clearly shows where the designed method shines: It

Table 1. Characteristics of the test data sets used in the experiments.

	German land cover	European land cover
Edges	1564	6635
Faces	525	2471
Neighbours per face		
Average	5.6	4.5
Median	5	3
Maximum	27	439*
Feature classes	12	29

*In this data set, some polygons have a relatively large extent (rivers) or have a lot of islands: this explains the high maximum number of neighbours.

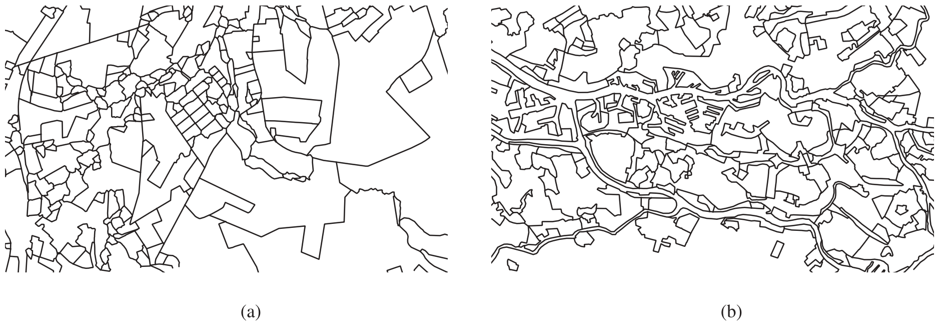


Figure 15. Clips from both test data sets. German land-cover data set. (b) European land-cover data set.

can handle fixed boundaries and create in a robust manner connectors to neighbouring areas (making it possible to create a new clean planar partition after the weighted split).

Section 1 mentioned that constructing data for the GAP data structure is an offline process, preparing the data for online use, for example, in a web environment. In every iteration of this batch process, the least important face is replaced by the result of a generalization operator (either a merge or, now with the availability of SplitArea, a split operation). At the end of every iteration, the obtained topological chains are put back into the planar area partitioning. From the topological chains, the polygon geometry is obtained to calculate the new area size of the neighbouring faces. In our experiment, no failures were found in reconstructing the geometry – this is an indication that all topological references were correctly set by our implementation of SplitArea.

5.1. *Evaluating splits and improving balance*

We first ran SplitArea in unweighted mode on the German land-cover data set. In this case, the attraction values are only determined by the boundary length between the splittee and its neighbours (cf. Equation 1, for which in this case, compatibility values and weights were set to 1, thus effectively only taking the length into account). For each split, the error per neighbour was recorded, according to the methodology described in Section 3.2 and post-processed into the total error value per split. Figure 16 shows a histogram and the spatial distribution of the total errors being made for all splits while building the hierarchical area-partitioning structure for this data set.

The errors stored per split allowed us to investigate which input was the basis for the splits with the highest error. We were most interested in what causes the errors in the tail of the histogram. The larger circles on the map in Figure 16(b) show those errors. It appeared that three types of input resulted in large differences between what we set as balanced target and what the algorithm obtained:

(1) The splittee is completely surrounded at one side by another face, which therefore devours the splittee and leaves no part for the other neighbour at all (e.g. Figure 17(a)).

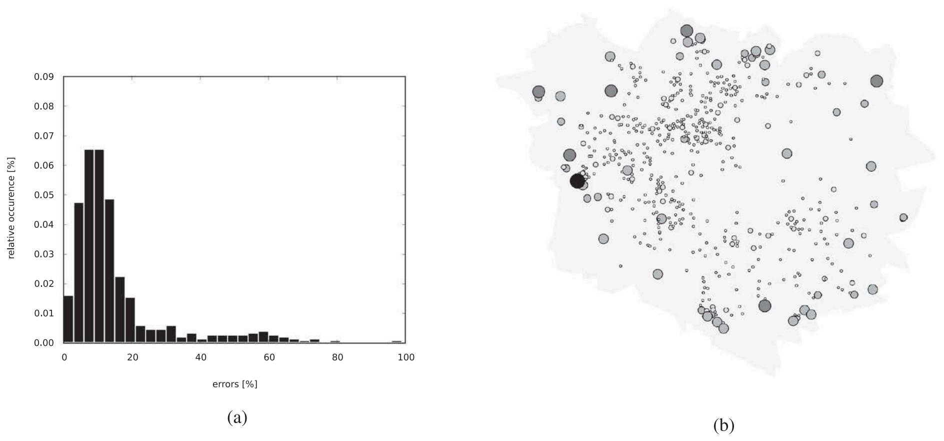


Figure 16. Results of first, unweighted run of SplitArea. (a) Histogram of the errors occurring while splitting faces. An error is the difference between the expected and actual share obtained by a neighbouring face. (b) Distribution of the errors over the domain. The larger and darker a circle is, the larger the total error in the outcome of the performed split is.

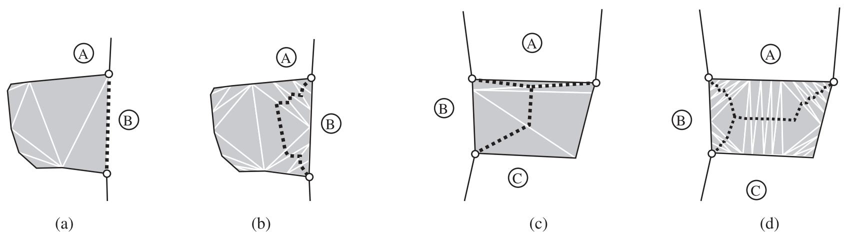
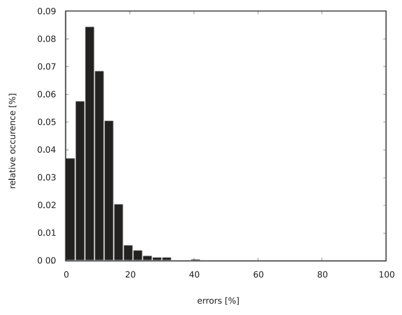


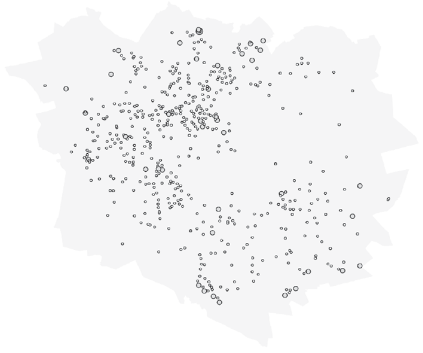
Figure 17. (a) & (b): Face A completely surrounds the splittee and gobbles it up. (c) & (d): On the boundary of the splittee, the vertices are distributed unevenly, leading to small triangles in front of face A: these triangles prevent face A from getting its balanced share of the splittee. (a) Completely surrounded splittee, face A gets all of the splittee. (b) Improved split. (c) Unevenly distributed vertices prevent face A from getting a share. (d) Improved split.

(2) A splittee having unevenly distributed vertices in its boundary, preventing one of the neighbours from getting a balanced share (thin and long triangles will appear at one side, preventing a face at this side to get a fair amount of the triangles on the interior of the splittee). See Figure 17(c) for an example.
(3) Faces at the border of the domain will give a share to the universe (i.e. we put zero for the expected share of the universe, but it will still get a piece).

To improve on the first two cases, we came up with the following strategy: first, delete the vertices that are unevenly distributed by simplifying the boundaries using a small tolerance value, and second, densify the boundaries, by placing new vertices regularly into the simplified boundaries. For simplification, we used an approach that does not introduce any topological errors, based on Kulik *et al*. (2005), and as a tolerance value we used twice the smallest segment length that is present in all the boundary edges of the splittee.



(a)



(b)

Figure 18. Results of a weighted run of SplitArea, using zero weights (with simplification and densification) to fixate the border of the domain. (a) Total errors made. (b) Distribution of errors over the domain. Errors at the border of the domain have been solved.

Figure 17(b) and (d) shows the result of two splits that clearly benefit from the strategy of simplifying and densifying. Overall, the total amount of error did decrease and it is evident that some of the larger errors did go away. However, the errors at the boundary of the domain can only be solved by fixing the borders of the splittee that are incident to the universe. Thus, to prevent the universe face from 'eating' objects, we placed zero weights on the edges incident to the universe. The result of this run is shown in the histogram in Figure 18, which is clearly the best result we obtained so far.

The problem of the algorithm not being able to get an even more balanced split (an 'error-free' result would result in a histogram with only one bar adjacent to the *y*-axis) is to be found in the limits imposed by the triangulation. Using a triangulation has advantages: it brings good control over maintaining correct topology, enables relatively easy computation of an 'approximate' skeleton and good and robust (triangulation) implementations[3] are freely available. However, the densification step that was shown to be necessary also shows that our solution space for finding the balanced split is limited by the way the triangles are placed based on the spatial configuration of the input or the spatial structure of the configuration as a whole. Smogavec and Žalik (2012) show that by inserting a number of Steiner points into the triangulation, the skeleton approximates more closely the medial axis. Their technique could improve the spatial configuration of the triangulation, which would allow us to get a better balanced split.

Another option is to switch to a different technique to obtain the segments of the skeleton. We considered using cartogram techniques (Dougenik *et al*. 1985) to obtain areas which result in areas sized exactly as specified. This has a difficulty that the connection to external chains is not guaranteed to be maintained automatically, and this is one of our hard requirements. We therefore investigated the use of the straight skeleton (Haunert and Sester 2008). Figure 19 illustrates that the straight skeleton algorithm produces a skeleton structure that touches every vertex on the boundary of the splittee (so connection between external chains is guaranteed; however, there is no possibility to choose connectors as with SplitArea).

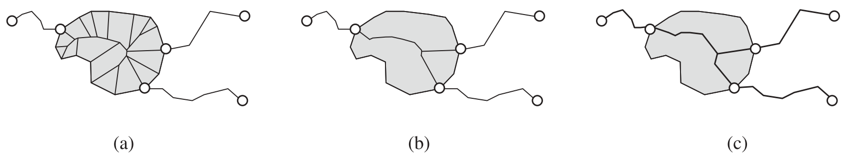


Figure 19. Comparing the straight skeleton, splitting the face of Figure 3, with the result of SplitArea. Whereas with SplitArea it is usually necessary to add additional connectors (cf. Figure 6(b)), the straight skeleton has connectors to all vertices and all unwanted branches need to be pruned. (a) Split using straight skeleton, including all branches. (b) Straight skeleton after pruning. (c) Split using SplitArea.

To fit the straight skeleton algorithm in our process flow (cf. Section 3), we replaced steps 1–4 with the straight skeleton algorithm[4]. After producing the skeleton segments, we construct a graph with skeleton segments and external chains and after labelling sides of the edges of this graph, we prune branches having the same label on both sides. Note that there is no robust straight skeleton implementation available that can generate arbitrarily weighted skeletons, although in theory this has been described (Biedl *et al.* 2015).

Using the same German land-cover data set as before, we ran both SplitArea as well as the straight skeleton algorithm. Again, no weights were used, and in both cases, the original boundaries were used as input (no simplification/densification was applied). During the process, we captured the errors made in division of the area of the splittee. Our results showed that the straight skeleton achieved somewhat fairer splits compared to SplitArea. We found that the average error made in appointing shares for the splittee is 2.7% when using the straight skeleton and 4.7% using the SplitArea method. Furthermore, Figure 20 illustrates the visual differences after splitting 480 area objects. From these images, it is clear that the straight skeleton produced somewhat smoother boundaries, but the overall shape of the areas is alike. This experiment showed that an alternative skeletonization technique can also fit with the algorithm as sketched in Section 3. Although the errors introduced are slightly smaller for the straight skeleton, engineering a robust, weighted straight skeleton implementation that can also deal with holes in the input polygons is not straightforward, as many degeneracies can occur (Kelly 2013).

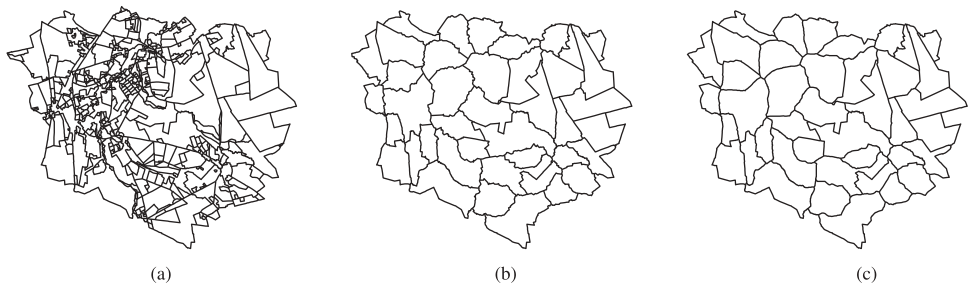


Figure 20. SplitArea versus straight skeleton after 480 splits. Note that in this case, no densification or line simplification has been applied. (a) German land-cover data set. (b) SplitArea. (c) Straight skeleton.

5.2. Effects for the topological vario-scale data structure

As the split area operation is new in the context of a hierarchical data structure (tGAP), it is important to assess the impact, both with respect to quantitative aspects (number of primitives) and visual impact. To compare the number of topological chains and faces that are the result of SplitArea with the number of primitives stored in the classic tGAP structure, we ran another experiment. In the classic tGAP structure, only a merge operation was available and duplicate edge rows are avoided as much as possible, cf. Meijers *et al.* (2009). For both test data sets, a classic tGAP structure was built, as well as a structure in which we applied SplitArea as the only generalization operation (without weights set on the input perimeter chains). Furthermore, we input for the compatibility values a matrix filled only with ones, which means that the merging and splitting operations are purely driven by geometric criteria (area and boundary length), but thematic attributes (i.e. looking at compatibility of feature classes) are not taken into account.

Table 2 shows the resulting number of topological chains (edges) and faces that are stored in the tGAP structure, depending on which generalization operation is applied. As our tGAP data structure is intended for a client–server setting with in-between (Internet) connections, it is important to return a proper number of edges and faces per generated map as well as a not-too-large number of vertices (for the edges), as more data means more communication time and possibly slower applications. The second row in Table 2 shows the result of applying only the merge operation, while the third row shows the results of applying SplitArea. In the case of applying a merge operation, a new face (having a new identifier) was introduced for the face that replaces the two old faces being merged. Therefore, the number of face rows is exactly 2× the original number of faces −1. When applying only the split operation, we also get one face less per operation, so we can conclude that the total number of faces could be equivalent to the number of faces as a result when merging. However, the difference of the number of split and of merged faces is because faces played a role in the split operation of one of their neighbours (not because they were split themselves) and were stored in the tGAP structure.

When looking at the number of edges, new edge records are only stored when these edges are merged; for more details, see Meijers *et al.* (2009). In case of applying SplitArea, the lengthened external chains, together with new edge geometries as result from the split, are stored in the tGAP structure. All in all, the number of primitives to be stored *will* be higher with SplitArea than when applying a merge operation, mainly because new boundary geometry is generated.

Figure 21 shows two map series as result of applying the two generalization operations. Although the differences between the two map series are subtle, they give an

Table 2. Empirical results: comparing merge operation with split operation.

		German land cover		European land cover	
Original	Edges	1564		6635	
	Faces	525		2471	
Merged	Edges	2533	1.6×	9795	1.5×
	Faces	1049	2×	4941	2×
Split	Edges	4913	3.1×	15,535	2.3×
	Faces	2498	4.8×	8840	3.5×

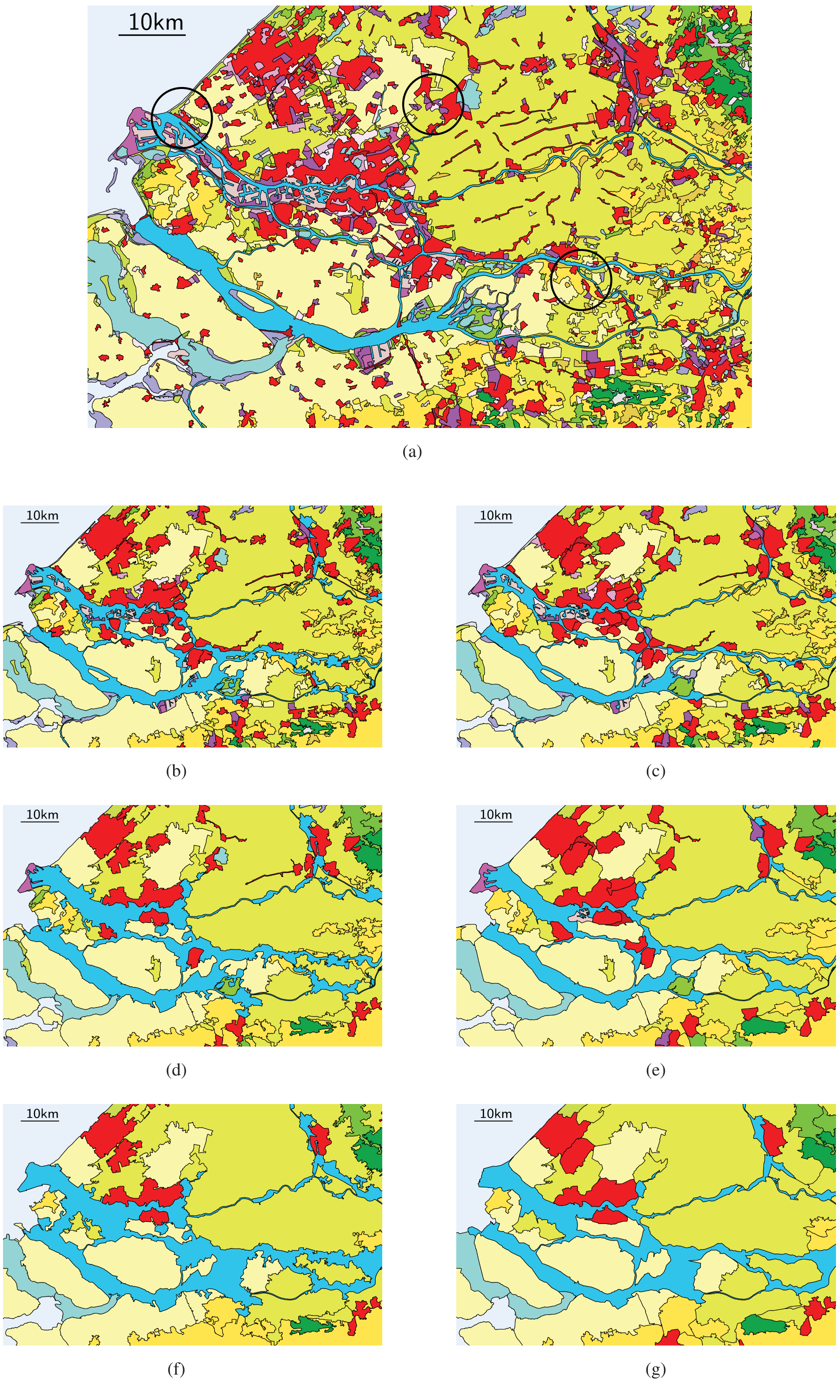


Figure 21. A series of maps derived from the tGAP structure. The series illustrate the (subtle) differences between the merge and split operations. Note that the maps with lesser areas should be drawn smaller and smaller. (a) Most detailed representation of European land-cover data set (1590 areas). The circles indicate regions where the merge and split approaches give notable different result. (b) Merge I (360 areas remaining). (c) Split I (360 areas remaining). (d) Merge II (120 areas remaining). (e) Split II (120 areas remaining). (f) Merge III (60 areas remaining). (g) Split III (60 areas remaining).

impression of what can be accomplished with both operations (without tweaking compatibility values at all). Merging leaves original boundaries of objects untouched, while SplitArea introduces new geometries. Because of this, the merge result seems to be a more 'all-or-nothing approach', especially when compared to the split operation (see urban areas in the Northwest part of the maps). However, in this experiment, the merge operation is hindered by the fact that the compatibility values were not tweaked much (and only were based on geometric criteria). Therefore, in some cases, land features and water features are merged inappropriately. Therefore, when making the generalization process more optimal, it should be considered to (1) tweak the compatibility matrix (see Hofman *et al*. 2008 for some investigation on how to obtain proper values), (2) investigate how to determine when to apply the split and when the merge operation (i.e. take more thematic attributes of the polygons into account) and (3) represent linear features explicitly in the structure (then linear networks, as often found in European topographic data, can play a role in generalizing the area partition as input for the tGAP structure).

6. Conclusion and future work

The algorithm presented, SplitArea, is a useful tool to perform a split operation between neighbouring faces or to obtain a thinned representation of a face. It was shown that the algorithm can be modified to handle weighted and unmovable topological chains as input, making a weighted split possible. Based on the weights, the attractiveness of individual faces can be steered and measured (an a priori goal can be set how to fairly divide a face). As a tool to use in the context of the tGAP data structure, SplitArea opens new possibilities for fine-tuning the generalization process: it is now possible to choose when to split (e.g. in case of linear road or water area features) or merge (other features). We have not yet implemented this choice of which operation to apply based on classification and thematic attribute knowledge in our generalization process, as it is known that orchestration of generalization operations is an important but not an easy task (Brassel and Weibel 1988, McMaster and Shea 1992, Stoter *et al*. 2013). For this purpose, both shape measures (e.g. perimeter vs area, or more advanced ones that determine vertex densities in the outline of the shapes) and feature classification (additional knowledge) can be used.

One idea we have for further tuning the algorithm itself is as follows. Section 4 introduced the weighted split algorithm: the principle behind this weighted split is that the location of the 'skeleton' should be pushed away for neighbours that should get more area (compatible or attractive neighbours) and pulled closer for neighbours that should get less area (incompatible or non-attractive neighbours). This works well for situations where neighbours are on opposite sides of the object. However, for making the 'skeleton' fit into the existing configuration, we needed to create connectors to external chains using existing unconstrained triangulation edges of which there is always at least one such edge, but there may be more. The choice of which connector to use does influence the quality of the weighted split when assigning fair area shares to the two objects left and right from the external chain. In our current implementation, we do not take into consideration the relative attractiveness of the two adjacent neighbours (at both sides of the external chain). Of course we have limited choices in the triangulation, but a fairer split would be to have a connector which is closest to the ratio of the

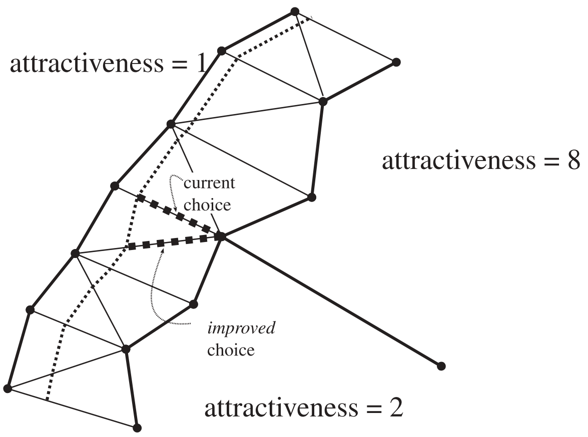


Figure 22. Influence of connector for fair split: weighted skeleton shown within splittee with dotted lines (fair based on weights at opposite site of splittee) and current connector depicted with short dashed line. A better connector, depicted with the long dashed line (based on the weights of adjacent neighbours at same side of the splittee).

weights (α_1:α_2 = weight$_1$:weight$_2$), where α_1 and α_2 are the angles (shares) of the 'pie' assigned to the two neighbours implied by the connector (Figure 22 presents an illustration). In addition, we could also consider adding a new connector edge at exactly the right angle (and not reusing an existing unconstrained triangulation edge). In this case, we have to be very careful not to introduce topology errors.

Another idea is to improve our metric, which is mostly based on the shares of the area size and does not include how the shape of the neighbouring boundaries changes. To obtain a good generalization, other shape aspects are also relevant, for example, the smoothness of the resulting line work, see straight skeleton experiment, or vertex density in the resulting line work, which could be changed by applying line simplification as post-processing step. These extra shape criteria somehow need to find their way into the defined metric. This formalization (development of quality measures) could also take place for other generalization operators. Perhaps these metrics may even become more formal data product specifications in general (and then are less related to the generalization process). Such quantifiable measures subsequently may be used in data product specifications (most likely with scale dependent values).

Furthermore, the algorithm brings other research questions. First, an unsolved question is how linear features can be represented as such in the tGAP structure, seeing the split operation now as a collapse operation (generating a linear representation for what first was modelled as an area feature) and then maintaining these edges as features (thus taking care of linear network connectivity, while surrounding area features are merged).

Secondly, SplitArea could probably be applied to *parts* of long and narrow polygons belonging to natural land-cover classes. Then it is necessary to define what these parts of the whole are. First computing the skeleton and then defining the width of the original polygon could probably help here for partly collapsing features. Morphological operators such as opening and closing could be helpful as well to identify those thin parts of the area features. Thirdly, for the more extreme generalizations, our approach leads to too many vertices still being kept. An idea under study is that the triangulation

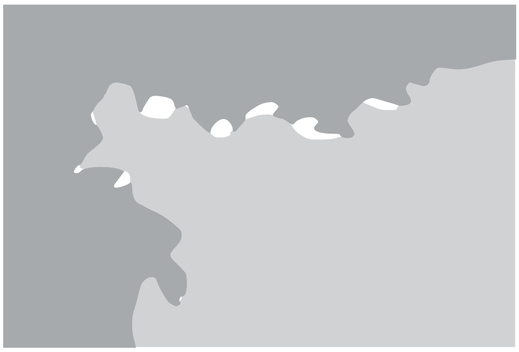


Figure 23. Alternative application of SplitArea: horizontal conflation. Two cross-border data sets have gaps (white areas) between them. SplitArea can be used to split those gaps. Figure taken from Ledoux and Arroyo Ohori (2011).

can be used to obtain a simplification of the new boundaries, which does not introduce any topological errors in the planar partition (the new skeleton vertices are only allowed to slide over the unconstrained edges of the triangulation – not introducing any errors). Because the main application of the tGAP data structure is in a web environment, the number of vertices in the output should be kept low.

In rather different contexts, SplitArea may be useful as well:

- When matching the boundaries of two cross-border data sets, also known as the edge-matching problem (Beard and Chrisman 1988, Chrisman 1990, Ledoux and Arroyo Ohori 2011, Ohori *et al*. 2012), the algorithm could be applied to allow splitting of slivers (overlaps or gaps that exist between the two data sets, cf. Figure 23). The metric of fairness we defined then allows to judge how fair the obtained solution is.
- Determine 'safe' path for a robot between hazardous areas, where the robot should stay away from the hazardous areas and not all areas are equally dangerous.
- Determine a 'playground area' when displacing area objects. After having selected a set of area objects, the space between these objects can be divided, weighting the space according to the importance of objects (giving more space to landmark objects for example).
- Reducing the number of tracks in a railway yard by triangulating the space between the track lines. New track lines can be generated more along the hull of the yard for outer two tracks (where weights set to zero), while new lines on the interior of the yard can be generated in between original lines.

Notes

1. Normally, the definition of the *skeleton* is a thin version of a shape that is equidistant to its boundaries (the so-called topological skeleton). In this paper, we use the term skeleton also for variants that approximate this thin version and is thus not necessarily equidistant to the boundaries (as for our application, an equidistant approximation is not wanted in all cases).

2. The source code of the implementation is available upon request from the first author.
3. For example, the triangulation from the CGAL project (http://www.cgal.org).
4. We used the implementation available in CGAL.

Disclosure statement

No potential conflict of interest was reported by the authors.

References

Ai, T. and Van Oosterom, P., 2002. Gap-tree extensions based on skeletons. *In*: D. Richardson and P. Van Oosterom, eds. *Advances in spatial data handling, 10th international symposium on spatial data handling*. Berlin: Springer-Verlag, 501–513. doi:10.1007/978-3-642-56094-1

Attali, D., Baja, G., and Thiel, E., 1995, Pruning discrete and semicontinuous skeletons. *In*: C. Braccini, L. DeFloriani, and G. Vernazza, eds. *Image Analysis and Processing, Springer Berlin/ Heidelberg, Lecture Notes in Computer Science*. Vol. 974. Berlin: Springer, 488–493.

Bader, M. and Weibel, R., 1997. Detecting and resolving size and proximity conflicts in the generalization of polygonal maps [online]. *In*: L. Ottoson, ed. *Proceedings of the 18th international cartographic conference*, 23–27 June Stockholm, 1525–1532. Available from: http://icaci.org/files/documents/ICC_proceedings/ICC1997/icc1997_volume3_part2.pdf

Bai, X. and Latecki, L.J., 2007, Discrete skeleton evolution. *In*: A.L. Yuille, *et al*., eds. *Energy minimization methods in computer vision and pattern recognition, Lecture Notes in Computer Science*. Vol. 4679. Berlin: Springer, 362–374.

Baumgart, B.G., 1975. A polyhedron representation for computer vision. *In*: *AFIPS '75: proceedings of the may 19-22, 1975, national computer conference and exposition*. New York, NY: ACM, 589–596. doi:10.1145/1499949.1500071

Beard, K.M. and Chrisman, N.R., 1988, Zipper: a localized approach to edgematching. *Cartography and Geographic Information Science*, 15 (2), 163–172. doi:10.1559/152304088783887071

Biedl, T., *et al*., 2015, Weighted straight skeletons in the plane. *Computational Geometry: Theory and Applications*, 48 (2), 120–133. doi:10.1016/j.comgeo.2014.08.006

Blum, H., 1967, A transformation for extracting new descriptors of shape. *Models for the Perception of Speech and Visual Form*, 19 (5), 362–380.

Brassel, K.E. and Weibel, R., 1988, A review and conceptual framework of automated map generalization. *International Journal of Geographical Information Systems*, 2 (3), 229–244. doi:10.1080/02693798808927898

Chrisman, N.R., 1990, Deficiencies of sheets and tiles: building sheetless databases. *International Journal of Geographical Information Systems*, 4 (2), 157–167. doi:10.1080/02693799008941537

Dougenik, J.A., Chrisman, N.R., and Niemeyer, D.R., 1985, An algorithm to construct continuous area cartograms. *The Professional Geographer*, 37 (1), 75–81. doi:10.1111/j.0033-0124.1985.00075.x

Gold, C. and Snoeyink, J., 2001, A one-step crust and skeleton extraction algorithm. *Algorithmica*, 30 (2), 144–163. doi:10.1007/s00453-001-0014-x

Haunert, J.-H. and Sester, M., 2008, Area collapse and road centerlines based on straight skeletons. *GeoInformatica*, 12 (2), 169–191. doi:10.1007/s10707-007-0028-x

Haunert, J.-H. and Wolff, A., 2010, Area aggregation in map generalisation by mixed-integer programming. *International Journal of Geographical Information Science*, 24 (12), 1871–1897. doi:10.1080/13658810903401008

Hofman, A., 2008. *Developing a vario-scale IMGeo using the constrained tGAP structure*. Thesis (Master's). TU Delft.

Hofman, A., *et al*., 2008. Using the constrained tgap for generalisation of imgeo to top10nl model [online]. *In*: W. Mackaness and S. Mustière, eds. *Proceedings of the ICA commission on map generalisation and multiple representation*. ICA Commission on Generalisation and Multiple

Representation, 1–23. Available from: http://generalisation.icaci.org/images/files/workshop/workshop2008/27_Hofman_et_al.pdf

Jones, C.B., Bundy, G.L., and Ware, M.J., 1995, Map generalization with a triangulated data structure. *Cartography and Geographic Information Science*, 22 (4), 317–331.

Kelly, T., 2013. *Unwritten procedural modeling with the straight skeleton*. Thesis (PhD). University of Glasgow.

Kieler, B., *et al*., 2009. Matching river datasets of different scales. In: M. Sester, L. Bernard, and V. Paelke, eds. *Advances in GISCIENCE, lecture notes in geoinformation and cartography*. Berlin: Springer, 135–154.

Kulik, L., Duckham, M., and Egenhofer, M., 2005, Ontology-driven map generalization. *Journal of Visual Languages & Computing*, 16 (3), 245–267. doi:10.1016/j.jvlc.2005.02.001

Ledoux, H. and Arroyo Ohori, K., 2011. Edge-matching polygons with a constrained triangulation [online]. *In*: *Proceedings of symposium GIS Ostrava 2011*, 24–26 January Ostrava. Available from: https://3d.bk.tudelft.nl/hledoux/pdfs/11_ostrava.pdf

Liu, Y. and Snoeyink, J., 2008, Faraway point: a sentinel point for delaunay computation. *International Journal of Computational Geometry & Applications*, 18 (4), 343–355. doi:10.1142/S0218195908002660

McAllister, M. and Snoeyink, J., 2000. Medial axis generalization of river networks. *Cartography and Geographic Information Science*, 27, 129–138. doi:10.1559/152304000783547966

McMaster, R.B. and Shea, K.S., 1992. *Generalization in digital cartography*. Washington, DC: Association of American Geographers.

Meijers, M., Van Oosterom, P., and Quak, W., 2009. A storage and transfer efficient data structure for variable scale vector data. *In*: M. Sester, L. Bernard, and V. Paelke, eds. *Advances in GIScience, Springer Berlin Heidelberg, lecture notes in geoinformation and cartography*. Berlin: Springer, 345–367.

Ohori, K.A., Ledoux, H., and Meijers, M., 2012, Validation and automatic repair of planar partitions using a constrained triangulation. *Photogrammetrie - Fernerkundung - Geoinformation*, 2012 (5), 613–630. doi:10.1127/1432-8364/2012/0143

Regnauld, N. and Mackaness, W.A., 2006, Creating a hydrographic network from its cartographic representation: a case study using Ordnance survey MasterMap data. *International Journal of Geographical Information Science*, 20 (6), 611–631. doi:10.1080/13658810600607402

Schwering, A., 2008, Approaches to semantic similarity measurement for geo-spatial data: a survey. *Transactions in GIS*, 12 (1), 5–29. doi:10.1111/tgis.2008.12.issue-1

Shewchuk, J.R., 2008, General-dimensional constrained Delaunay and constrained regular triangulations, I: combinatorial properties. *Discrete & Computational Geometry*, 39 (1–3), 580–637. doi:10.1007/s00454-008-9060-3

Shewchuk, J.R. and Brown, B.C., 2015, Fast segment insertion and incremental construction of constrained Delaunay triangulations. *Computational Geometry*, 48 (8), 554–574. doi:10.1016/j.comgeo.2015.04.006

Smogavec, G. and Žalik, B., 2012, A fast algorithm for constructing approximate medial axis of polygons, using Steiner points. *Advances in Engineering Software*, 52 (1), 1–9. doi:10.1016/j.advengsoft.2012.05.006

Stoter, J., *et al*., 2013, Fully automated generalization of a 1:50k map from 1:10k data. *Cartography and Geographic Information Science*, 0 (0), 1–13.

Szombara, S., 2013. Unambiguous collapse operator of digital cartographic generalisation process [online]. *In*: D. Burghardt, ed. *Proceedings of the 16th ICA generalisation workshop*. ICA Commission on Generalisation and Multiple Representation, 1–10. Available from: http://generalisation.icaci.org/images/files/workshop/workshop2013/genemappro2013_submission_20.pdf

Uitermark, H., Vogels, A., and Van Oosterom, P., 1999. Semantic and geometric aspects of integrating road networks. *In*: A. Včkovski, K.E. Brassel, and H.-J. Schek, eds. *INTEROP '99: proceedings of the second international conference on interoperating geographic information systems*. Vol. 1580. London: Springer-Verlag, 177–188.

van Oosterom, P., 1993. The GAP-tree, an approach to "On-the-Fly"' map generalization of an area partitioning [online]. In: J.-C. Mueller, J.P. Lagrange, and R. Weibel, eds. *GIS and generalization,* *methodology and practice, GISDATA specialist meeting on generalization*, 15–19 December Compienge. London: Taylor & Francis, 120–132, 1995. Available from: https://www.crcpress.com/GIS-And-Generalisation-Methodology-And-Practice/Lagrange-Weibel-Muller/9780748403196

Van Oosterom, P., 2005. Variable-scale topological data structures suitable for progressive data transfer: the GAP-face tree and GAP-edge forest. *Cartography and Geographic Information Science*, 32, 331–346. doi:10.1559/152304005775194782

Van Putten, J. and Van Oosterom, P., 1998. New results with generalized area partitionings. *In*: T.K. Poiker and N. Chrisman, eds. Proceedings: *8th international symposium on spatial data handling*. Burnaby, BC: International Geographical Union, Geographic Information Science Study Group, 485–495.

Van Smaalen, J., 2003. *Automated aggregation of geographic objects: a new approach to the conceptual generalisation of geographic databases*. Thesis (PhD). Wageningen University.

Vermeij, M., *et al*., 2003. Storing and using scale-less topological data efficiently in a client-server DBMS environment. *In*: *Proceedings of the 7th International Conference on GeoComputation* [online], 8–10 September University of Southampton, Southampton. GeoComputation CD-ROM. Available from: http://www.geocomputation.org/2003/Papers/Vermeij_Paper.pdf.

Worboys, M.F. and Duckham, M., 2004. *GIS: a computing perspective* [online]. 2nd ed. Boca Raton, FL: CRC Press. Available from: https://www.crcpress.com/GIS-A-Computing-Perspective-Second-Edition/Worboys-Duckham/9780415283755

Yang, J., *et al*., 2013, Land use patch generalization based on semantic priority. *Abstract and Applied Analysis*, 2013, 1–8.

Yaolin, L., *et al*., 2003, Categorical database generalization. *Geospatial Information Science*, 6 (4), 1–9. doi:10.1007/BF02826943