Accepted at International Symposium on Spatial Data Handling, SDH'98 Vancouver, Canada, 12-15 July 1998

New results with Generalized Area Partitionings*

Judith van Putten†
Department of Computer Science, Utrecht University
P.O. Box 80.089, 3508 TB Utrecht, The Netherlands.
Email: judith@cs.ruu.nl
and
Peter van Oosterom‡
Cadastre Netherlands, Company Staff
P.O. Box 9046, 7300 GH Apeldoorn, The Netherlands.
Email: oosterom@kadaster.nl

This paper presents the on-the-fly generalization results with the GAP-tree (Generalized Area Partitioning) implementation applied to two large scale topographic data sets: vGBKN 1:1.000 and Top10vector 1:10.000. Applying the GAP-tree, it is now possible to interactively browse through these spatial data sets at medium 1:25.000 and even small < 1:100.000 scales. In addition to the test results and implementation of standard GAP-tree, an extension is described based on adding parallel lines to the GAP-tree for linear features such as roads and waterways. This enables better, more smooth, enlargements of these features required at smaller scales. Further, a new method is given to build a GAP-tree representation for a very large data set based on a non-quadtree like divide-and-conquer technique.

Keywords: Generalization, area partitioning, spatial data structure, topography, internet/web map

1 Introduction

The advent of Geographic Information Systems (GISs) has changed the way people use maps or geographic data. The modern information systems enable browsing through geographic data sets by selecting (types of) objects and displaying them at different scales. However, just shrinking the objects when the user zooms out, will result in a poor map. This map would also be slow to produce, because it contains too much information. Not only should the objects be shrunken, but they should also be displayed with less detail (because of the smaller scale). Further, less significant objects should be omitted at all, because they do not serve the map at the displayed smaller scale. A simple solution is to store the map at different scales (or level of details). This would introduce redundancy with all related drawbacks: possible inconsistency and increased memory usage. Therefore, the geographic data should be stored in an integrated manner without redundancy, and if required, supported by a special data structure.

Detail levels are closely related to cartographic map generalization techniques. The concept of *on-the-fly* map generalization [14] is very different from the implementation approaches described in the paper of Müller et al.: *batch* and *interactive* generalization [6]. The term batch generalization is used for the process in which a computer gets an input data set and returns an output data set using algorithms, rules, or constraints [4] without the intervention of humans. This in contrast to interactive generalization ("amplified intelligence") in which the user interacts with the computer to produce a generalized map. On-the-fly map generalization does not produce a second data set, as

*A color PostScript version of this paper is available at `http://ooa.kadaster.nl` in the papers section.

†The research was performed while this author was at the Cadastre working on the project to obtain a MSc-degree from Utrecht University

‡Corresponding author

this would introduce *redundant* data. It tries to create a temporary generalization, e.g., to be displayed on the screen, from one detailed geographic database. The quick responses, required by the interactive users of a GIS, demand the application of specific data structures, because otherwise the generalization would be too slow for reasonable data sets. Besides being suited for map generalization, these data structures must also provide spatial properties; e.g., it must be possible to find all objects within a specified region efficiently. The name of these types of data structures is *reactive data structures* [11, 12, 13].

The GAP-tree, a reactive data structure suited for area partitionings, is described in Section 2 based on [14]. The implementation of the GAP-tree (Section 3) is evaluated with two data sets: vGBKN and Top10vector; see Section 4. Some improvements for the GAP-tree are given in Section 5. Finally, conclusions are given in Section 6 along with some indications of future work.

2 The GAP-tree

An area partitioning, where each point in the 2D domain belongs to exactly one of the area features, is often used as a map basis. Some problems occur when applying generalization techniques:

1. *Simplification:* Independent line generalization of the boundaries of two neighboring area features might result in ugly maps because the area features may now have overlaps and/or gaps. A solution for this problem is to use a topological data structure and apply line generalization to the common edges.
2. *Selection:* Leaving out an area will produce a map with a hole which is of course unacceptable. No obvious solution exists for this problem.

The gap introduced by leaving out the least important area feature must be filled again. The best results will be obtained by filling the gap with neighboring features. This can be easy in case of so called “islands”: the gap will be filled with the surrounding area, but a good choice may be more difficult in other situations. The least important area feature a has the lowest value for importance function; for example: $Importance(a) = priority_{ta} \times Area(a)$ where $priority_{ta}$ is the weight factor for the feature type of a. Neighbor b with the highest value for collapse function may fill the gap of feature a. An example collapse function is $Collapse(a, b) = compatibility_{ta,tb} \times Length(a, b)$ with $compatibility_{ta,tb}$ the compatibility between feature types of a and b, and $Length(a, b)$ the length of the shared boundary between a and b. The compatibility is related to closeness of two feature types of a and b in the feature classification hierarchy associated with the data set. By taking into account the spatial relationships, the importance and the compatibility of area features into account, an *area partitioning hierarchy* is created.

The polygonal area partitioning is usually stored in a topological data structure with nodes, edges, and faces. The process, described below, for producing an area partitioning hierarchy assumes such a topological data structure [1, 2, 5, 7]. The topological elements have the following attributes and relationships:

- a *node* (or 0-cell) contains its point and a list of references to incoming/outgoing edges sorted on the associated angle;
- an *edge* (or 1-cell) contains its polyline, length, references to the left and to the right faces, and references to the begin and to the end nodes;
- a *face* (or 2-cell) contains its area, and a list of sorted and signed references to edges forming the outer ring (a sequence of edges forming a closed loop) and possibly inner rings (related to islands);

This topological data structure contains some redundant information, enabling more efficient processing later on. After the topological data structure has been created, the following steps will produce a structure, called the *Generalized Area Partitioning* (GAP)-tree (more details in [14]), which stores the required hierarchy:

1. For each face a in the topological data structure an “unconnected node in the GAP-tree” is created and an insert in the priority queue is done based on the value of $Importance(a)$;
2. Select the least important area feature a. The priority queue supports the required operation *dequeue* (extract_min) to find this feature efficiently.

3. Use the topological data structure to find the neighbors of a and determine for every neighbor b, the list of shared edges and the *total* length of this common boundary $Length(a, b)$.

4. Fill the gap by selecting the neighbor b with the highest value of the function: $Collapse(a, b)$. More details w.r.t. steps 3 and 4 can be found in Subsection 3.1;

5. Store the polygon and other attributes of face a in its node in the GAP-tree and make a link in the tree from parent b to child a;

6. Add face a to face b, adjust the topological data structure (Subsection 3.2), recompute $Importance(b)$, and adjust the position of feature b in the priority queue based on the new importance value.

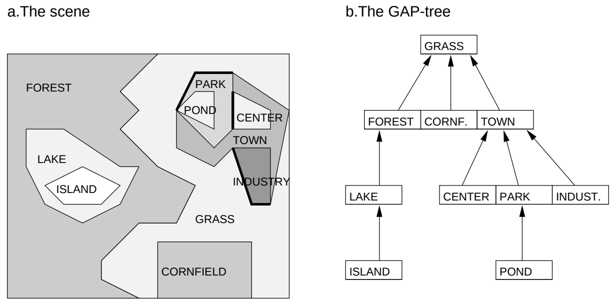


Fig. 1: The scene and the associated GAP-tree

Repeat the described steps 2–6 until only one huge area feature is left. The result is stored in the GAP-tree and the last area feature will form the root of the GAP-tree. During the interactive use in a GIS, the pre-computed GAP-tree is descended until the importance of the current node (area feature) is below the required importance level for a certain display operation. Figure 1.a shows a scene with a land-use map in the form of an area partitioning. In Figure 1.b the GAP-tree, as computed by the procedure described above, is displayed. The polygon is a real self-contained simple polygon with coordinates, and it is not a list of references to edges. The GAP-tree is a multi-way tree and not a binary tree. Some visual results of the on-the-fly generalization techniques with real data are displayed in Section 4.

3 Implementation

This section describes in more detail how to find best neighbor, including all shared edges with this neighbor (Subsection 3.1) and how to adjust the topological data structure when collapsing two area features (Subsection 3.2).

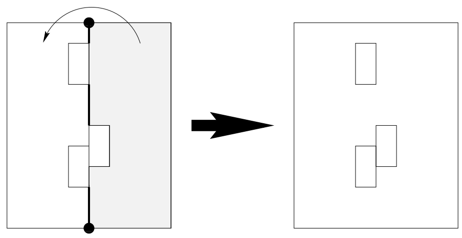


Fig. 2: New islands may arise when joining faces.

3.1 Finding the best neighbor (steps 3 and 4)

The least important face is delivered by the priority queue in step 2 and will be called face a; see Section 2. The boundary of a face is stored as a list of ordered edges, which includes both the outer ring and possible also inner ring(s) (enclosing islands). To find a neighbor of face a, one has to look at an edge on the boundary and determine which face is on the other side of this edge. So, when all edges on the boundary of a face are traversed, all neighbor faces can be found. It is possible that a neighbor has multiple edges shared; see Figure 2. It is important that all these edges are found before the collapse value is computed, because this is based on the total length of the shared boundary.

During the sequential traversal of the edges within a ring, an initially empty list L is created which keeps track of all neighbor faces in progress. That is, neighbors of which possibly not all shared edges may be found yet. Every time a new neighbor is found, it is added to the front of list L. In case neighbor b is found again, that is, b is already somewhere in list L, then all neighbors in list L before b are completely found. They can be processed and removed from list L. When all edges in a ring are handled there will still be one or more neighbor faces in list L, which can now also be processed. Processing a neighbor includes: calculating the collapse value (based on the total length of the shared edges and the compatibility of the feature types) and saving the neighbor with the highest collapse value. Note that the procedure described above does work for the outer ring (face a is a normal neighbor or face a is an island of parent face b) and also for the inner ring (face a is the parent of island face b).

Is was stated above that when neighbor b is found again, all neighbors in front of neighbor b in list L could be processed. It will now be proven that for all neighbors in list L before neighbor b, all the shared edges are known. Suppose the contrary. This means that there is a neighbor face c before face b in list L which is not completely found. This means that face c has shared edges between and after two shared edges of face b on the boundary of face a; see Figure 3. But this situation implies that the faces b and c intersect/cross each other, which is impossible for a planar topology. When the data set is correct this can not happen.
contradiction

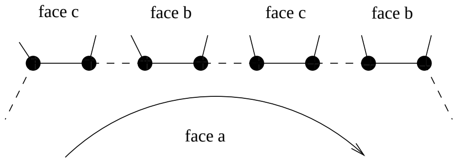
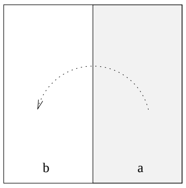


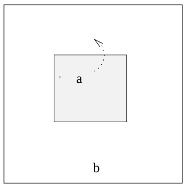
Fig. 3: Face a with impossible neighbors b and c.

3.2 Adjust the topological data structure (step 6)

How the least important face a is added to its best neighbor b is discussed in this subsection. Faces are represented as list of edges in the topology structure. In order to join two faces, two lists of edges (*edgelists*) have to be connected, the shared edges have to be removed and the new islands which may arise, have to be discovered. It depends on the situation which edgelists have to be connected. Three different situations can be distinguished. These situations are shown in Figures 4, 5 and 6 (all simple cases) and Figures 7, 8 and 9 (all complex cases). These different situations can be recognized by the use of a polygon-in-polygon test (or point-in-polygon with the centroid).

In the situation 'Normal neighbors' the edgelist of the outer ring of face a and the edgelist of the outer ring of face b have to be connected. That is,





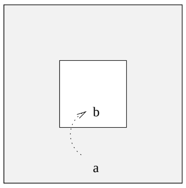
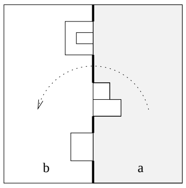
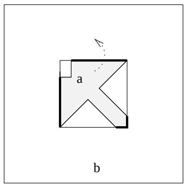


Fig. 4: Normal neighbors (1). Fig. 5: Island to parent (1). Fig. 6: Parent to island (1).





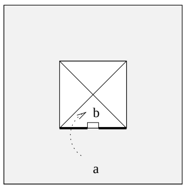


Fig. 7: Normal neighbors (2). Fig. 8: Island to parent (2). Fig. 9: Parent to island (2).

shared edges are removed and the remaining parts of the edgelists are coupled. In the situation 'Add island to parent' the edgelist of the outer ring of face a and the edgelist of the matching inner ring of face b have to be connected. In the situation 'Add parent to island' the edgelist of the matching inner ring of face a and the edgelist of the outer ring of face b have to be connected. Note that in these last two cases complete edgelists may be removed; see Figures 8 and 9. The way remaining parts of the two edgelists have to be connected is in all situations the same. A shared edge is part of both edgelists. The edges in front of and after the shared edge in the edgelists have to be connected; see Figure 10.

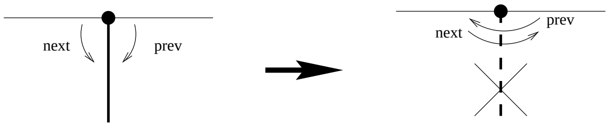


Fig. 10: Connecting edgelists.

Connecting the edgelists has to be done for all shared edges on both sides (nodes) of each shared edge. During this process new edgelists arise which form loops. A new edgelist can be found after a shared edge is handled. The list of shared edges EL may be handled forwards or backwards, but the ordering of the shared edges may not be changed. In case a shared edge e is not the first edge in list EL, then find a new edgelist at the node of shared edge e at the side where other shared edges are already handled; e.g. node a of edge $e2$ in Figure 11 assuming that shared edge $e1$

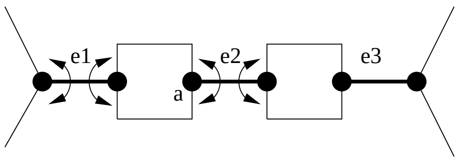


Fig. 11: Forming new (inner) rings.

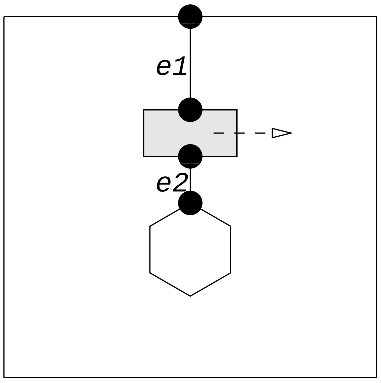


Fig. 12: New dangling edges.

is first processed. In case a shared edge e is the last edge in list EL (e.g. $e3$), then also find a new edgelist at the other side of this shared edge.

Two different kinds of edgelist, representing new rings of the boundary of the enlarged face, can be distinguished. Edgelists with clockwise orientation and edgelists with counter clockwise orientation. Only one edgelist with clockwise orientation will arise. This edgelist is the outer ring of the enlarged face. All the other edgelists will have counter clockwise orientation. These edgelists are new inner rings (enclosing islands) of the enlarged face.

When joining two faces it may be possible that new dangling edges arise, which have to be removed. That is, two succeeding edge references in a newly created edgelist refer to the same edge, but in opposite directions; see edges $e1$ and $e2$ in Figure 12. These edges have to be discovered and removed. Note that a new dangling edge can only arise when the original edge did already have the same face on both sides. These edges are not needed for representing islands in the planar partition, but may occur in the data set as linear features without a width; e.g. a fence. Instead of removing these edges during the creation of the GAP-tree, they could also have been removed in advance.

4 Results

Two different large scale topographic maps are are used to test the GAP-tree: GBKN (Subsection 4.1) maintained by the Cadastre and the TOP10vector (Subsection 4.2) maintained by the Topographic Service. Both maps are an accurate representation of the terrain without any displacement of features due to cartographic reasons. However, in TOP10vector some feature representations are simplified.

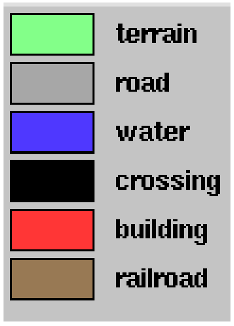


Fig. 13: Legend for the maps.

Table 1: GBKN type priority

name	color	code	priority
terrain	green	TRN	5
road	grey	WEG	80
water	blue	WTR	10
crossing	black	KNP	2
building	red	GBW	400
railroad	brown	SBN	20

4.1 Results with the GBKN

The large-scale base map of The Netherlands, known as GBKN, is usually produced by photogrammetric *stereo* plotting with field completion (especially for buildings). Its presentation scale is 1:1.000 and it has an almost nationwide coverage of buildings, roads and waterways. The accuracy of the GBKN is stated in terms of *relative* precision: in rural areas the relative precision between two well defined points should be better than $40\sqrt{2}cm$, in urban areas better than $20\sqrt{2}cm$ [8]. The GBKN is a line oriented map and has no area features. In the future also, topologically structured, area features will be represented in the map called vGBKN. A prototype vGBKN data set has been produced of the municipality of Zevenaar, and contains six main area feature types; see Table 1.

The generalization requirements for the vGBKN depend on the purpose of the generalized map; assume this is to visualize the clusters of urban area's and the connecting roads. So, crossings (low priority) may be added to roads. Further, assume that railroads and waterways are not important (low priority) and may be added to the terrain. In the *urban* territory it is required that:

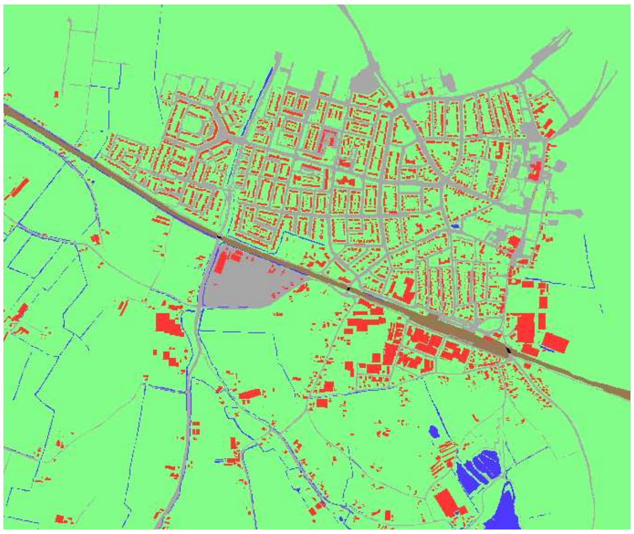


Fig. 14: vGBKN 1 (original map)

Fig. 15: vGBKN 2

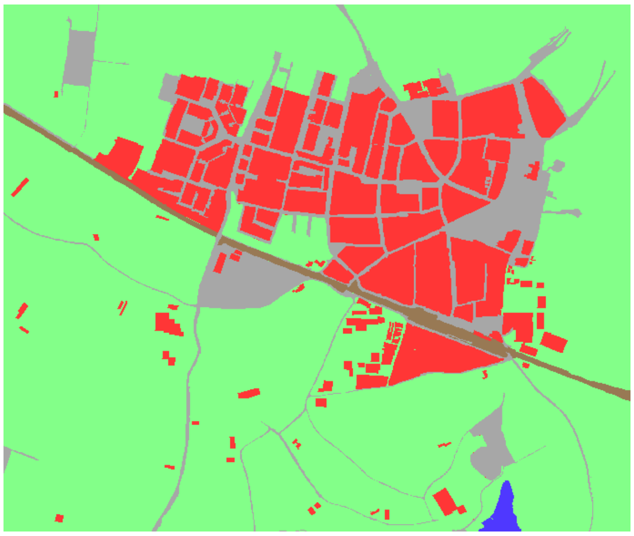


Fig. 16: vGBKN 3

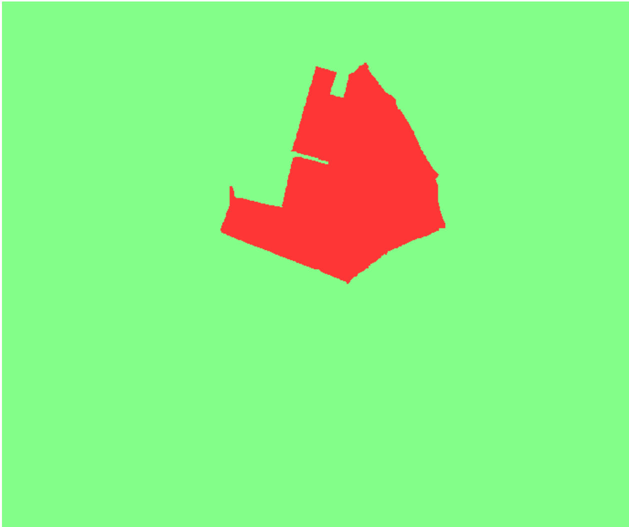


Fig. 17: vGBKN 4 (final map)

1. terrain should be added to buildings, progressively; 2. then roads should be added to enlarged buildings (first unimportant roads then important roads); 3. finally, one big building face should be left, possibly with a number of important roads; In the *rural* territory it is required that: 1. buildings should be added to the terrains; 2. terrains and the other kind of faces should be joined together; 3. roads must stay on the map for as long as possible; 4. finally, one big terrain face should be left.

In the GBKN data set there is no distinction between the grounds in urban territory and the terrains in rural area. The only distinction between these faces is the size of their area. With this in mind and after several iterations of experiments

Table 2: GBKN type compatibility (vertical: face to be removed, horizontal: face to be added to)

	TRN	WEG	WTR	KNP	GBW	SBN
TRN	1.0	0.1	0.4	0.1	0.7	0.1
WEG	0.2	1.0	0.1	0.9	0.2	0.005
WTR	0.6	0.005	1.0	0.005	0.005	0.005
KNP	0.5	0.9	0.1	1.0	0.005	0.8
GBW	0.9	0.005	0.1	0.005	1.0	0.005
SBN	0.5	0.005	0.1	0.8	0.005	1.0

with the importance and compatibility values, the Tables 1 and 2 are obtained (more details can be found in [16]). The visual results of the on-the-fly generalization of the vGBKN can be seen (all on the same presentation scale for good comparison) in Figures 14, 15, 16, and 17 by incrementally raising the minimum importance value. The relationship between the minimum importance value and the proper scale still has to be determined by further experiments. Pushing the GAP-tree by requiring very high importance values, may change the Gestalt, as in Figure 16. However at the target scale this region covers only 4 or 9 pixels on the screen, so the effect is not visible.

As the emphasis is on generalization of area partitionings with the GAP-tree, the line and points features have been omitted. In Figure 16 the roads in rural territory are not completely present. The only way to solve this is to increase the priority of roads. But to increase the priority of roads implies that roads get more important than the enlarged buildings. In fact, the roads in rural territory have to get more important, progressively. This may be achieved by enlarging the width of these roads, implying that also the area and the importance of these roads is increased progressively. A more detailed description of this idea can be found in Section 5.

4.2 Results with the TOP10vector

The TOP10vector is usually produced by photogrammetric *mono* plotting with field completion. Its presentation scale is 1:10.000 and it has an almost nationwide coverage of buildings, roads, waterways and many different terrain types. The accuracy of TOP10vector is stated in terms of *absolute* precision with relation to the national reference system: the location of points should be better then 2 meter. It is assumed that the generalization goal with the TOP10vector is the same as in the GBKN test; see Subsection 4.1. The TOP10vector has more detailed classifications for the different types of the terrain features.

In this data set there is a special classification for terrain in urban territory, called 'grounds'; see faintly red regions in Figure 18. The grounds can be added easily to the buildings in urban territory by giving them a low priority and high compatibility with buildings. This means that the other terrain features can have a higher priority value compared to the GBKN data set. Roads are distinguished in important roads, unimportant roads and cycle-tracks. This creates the opportunity to handle these three types of roads differently. Important roads can be maintained much longer than unimportant roads. This is done by giving the important roads a higher priority than the unimportant roads. Also important roads can be maintained in one part. This means that the important road faces have to be joined as much as possible. Therefore, the compatibility between an important road and all the other kind of faces have to be reduced, so it becomes more likely that important road faces will be joined with each other. In the GBKN data set this only occurred to a certain degree which is still desirable for the unimportant roads. Cycle-tracks get the lowest priority of all types of road and a high compatibility with those other types of road. This implies that the cycle-tracks will be selected before the other types of road and be added to these other types.

After several iterations of testing priority and compatibility tables, the results of GAP-tree generalization of the TOP10vector can be seen in Figures 18, 19, 20, and 21 (more details can be found in [16]). Note that is is still impossible to keep the roads on the highly generalized maps. Also for this data set, an improvement of the GAP-tree method is required; see Section 5.

5 Improvements to the GAP-tree

Until now results with the standard GAP-tree are presented. In this section two important improvements are described: parallel lines for linear features (Subsection 5.1) and handling very large data sets (Subsection 5.2.

5.1 Introducing parallel lines

In Section 4 was mentioned that the width of the roads should be enlarged progressively. This may be achieved by adding parallel lines to these 'linear' feature types, in the input data set. To enlarge the roads progressively one or more of these parallel lines have to be created for these roads on various distances; e.g. at d, $2d$, $4d$, and so on. So, smaller (less important) strips implied by the parallel lines are closer to the original road feature and will be added to the original feature early on. The number of parallel lines and the initial distance d are parameters for feature types (similar

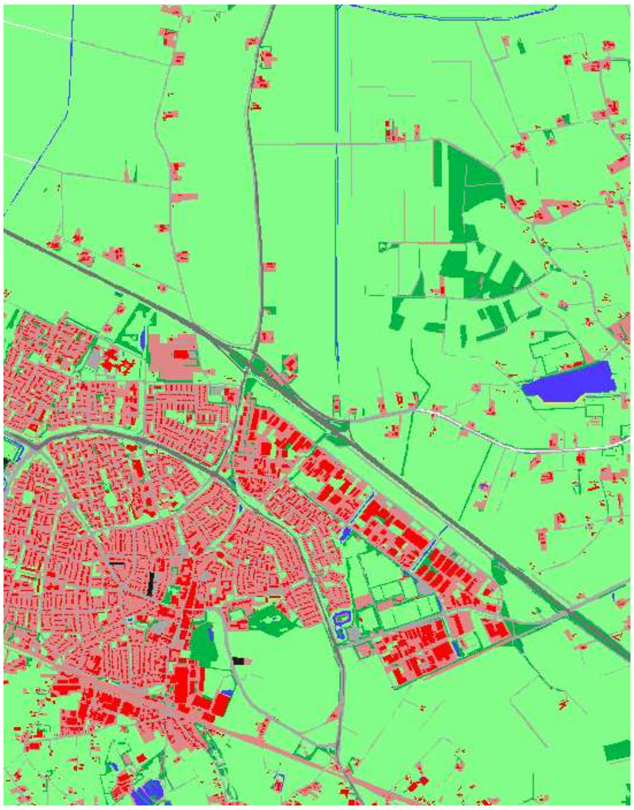


Fig. 18: Top10vector 1 (original map)

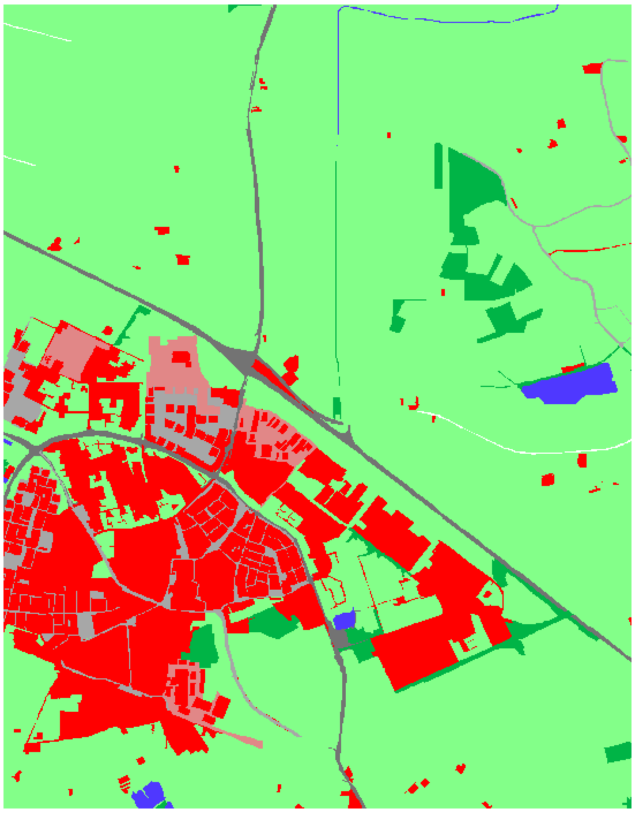


Fig. 19: Top10vector 2

to importance). Adding the strips one by one to the original face will enlarge it progressively.

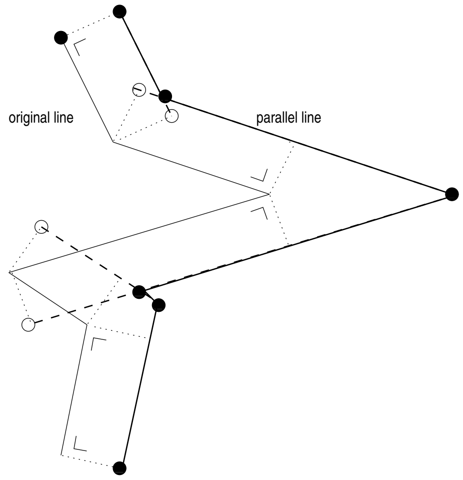


Fig. 22: Parallel line.

One way to construct a parallel line is to traverse all line segments of the polyline (edge). A parallel line segment can be created by moving the original line segment along its perpendicular. To make from two neighbor parallel line segments one parallel polyline an intersection point of the two parallel base lines has to be calculated. Figure 22 shows an example of a parallel polyline created this way. This can be done for all line segments, because two non-parallel lines always have an common intersection point (take care of 'exceptions').

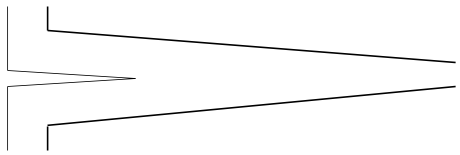


Fig. 23: Situations with very small angles.

When there are two line segments which share a very small angle, then the two parallel lines will have an intersection point which is situated very far away; see Figure 23. This can be avoided by a check on the distance between the intersection point of the original line segments (o) and the intersection point of the created line segments (p). When this distance is larger than a certain default value, replace p by another point closer to o which honors the default value. An example is shown is Figure 24.

In the GBKN data set parallel faces may be created for *roads*. These parallel faces should be joined with the roads progressively. Before the GAP-tree is built the created parallel faces will

Fig. 20: Top10vector 3

Fig. 21: Top10vector 4 (final map)

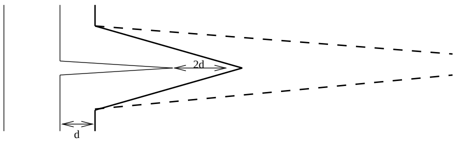


Fig. 24: Solution for a very small angle.

not be of the type road, but it will have the same type as the face (f) which used to be on the same place on the map, usually terrain. So face f is divided in a several disjoint faces of the same type depending on the number of parallel lines. The *Collapse* function should take care of the fact that these parallel faces should be joined with the corresponding road progressively, that is $Collapse(p, r) > Collapse(p, a)$, where face r is a road, face p is (part of) the parallel face which has been created for road r, and face a is face f minus face p. In the GBKN data set this face a will most likely be a terrain. One way to solve the problem is to make the compatibility between TRN-WEG larger than TRN-TRN. As this may seam counter intuitive, another solution to the problem could be to adjust the Collapse function in a way that also the neighbor priority are used to calculate the collapse value. For this purpose the Collapse function could be defined in the following way: $Collapse(a, b) = priority_b \times compatibility_{ta,tb} \times Length(a, b)$ When this Collapse function is used in the present example, then the parallel faces will be added to the roads. But what will happen with all the other (non-parallel) faces? Now all the non-parallel faces are in danger of being joined with the most important neighbor face instead of the most compatible neighbor face! Further tests are required here. It may be useful to make a distinction between a 'parallel' face and an 'ordinary' face, w.r.t. priority and compatibility values.

5.2 GAP-tree for very large data set

During the execution of the tests, we encountered 'machine limitation' when building a GAP-tree for a very large data set. The current algorithm is based on a main memory approach, before the result is stored again in the database. So, data sets are becoming too large to be processed at once by a GAP-tree construction program. The size of the data sets will increase even more if parallel lines of the previous subsection are introduced. A technique is required to process the data sets in smaller units, without affecting the final result too

Table 3: Number of elements per representation (number of coordinates within brackets)

	GBKN	Top10vector
#edges	8.700 (70.000)	22.500 (125.000)
#faces	4.700 (95.000)	12.000 (150.000)
#gaps	4.700 (120.000)	12.000 (220.000)

much. It is important to note that in the database, where the GAP-tree is stored, there is no significant overhead.

In Table 3 it is shown how many coordinates are stored in the edges (of the planar topology), how may coordinates would be stored in the polygons (in a representation without topology), and how many coordinates are stored in the GAP-tree representation. The overhead (redundant coordinates) of the GAP-tree is about 30% to 40% more more compared to the polygon based representation. One might expect that the number of coordinates in the polygon based representation would be nearly two times larger that the number of coordinates in the topology based edge representation. From Table 3 it can be observed that this is not the case. An explanation for this fact is that our edges have a relative few number of points per line and the begin and end point are included (introducing redundancy in nodes)[1]. Further, there are many houses in the data sets, which will result in many holes in the topology structure. In this case the edge based representation has even one coordinate more than the polygon based representation, which does not repeat the begin/end point.

However, due to the internal data structure in the program, there is a significant overhead during the construction of the GAP-tree. Assume that a $2^k * 2^k$ processing grid can be imposed on the domain with on the average $4n$ area features with their centroid inside a processing grid cell. These $4n$ area features are generalized with the GAP-tree to n features. Then this processing grid cell can be merged with it's 3 quadtree neighbors [9]. The larger processing grid cell will contain again $4n$ generalized area features on the average. The same steps can be repeated until there is only one processing grid cell left.

In the GBKN data set parallel faces may be created for *roads*. These Drawback of this quadtree method will be that two very similar features, close together, but on the other side of a major quadtree split line will not be joined until the last step. Therefore, Field-tree [3] is suggested as the split lines are shifted at every level in this 'tree'. An additional advantage of this divide-and-conquer method to build the GAP-tree, is that it may be easier to make local updates without recomputing the whole data set. The local change will not influence the neighbor grid cell processing. However, all levels above will still have to be recomputed.

6 Conclusion

The GAP-tree is a structure which can store all the generalizations in such a way that for every scale the matching generalizations can be determined, rapidly. In this project the GAP-trees were built for two different data sets: vGBKN and Top10vector. After setting the feature type priority and compatibility values, which turned out to be non trivial, the generalizations were performed completely automatically. To what extent the generalization requirements can be met depends on the way the features can be distinguished. The results with a more detailed classified data set are better than with a less detailed data set.

Tests with GAP-trees, with parallel lines for certain feature types, are planned for the near future. Better tools for setting feature type importance and compatibility between feature types are required, in order to easily produce GAP-trees for other tasks. Other data structure can be used in combination with the GAP-tree. The *Reactive-tree* may be used to efficiently store the GAP-tree in a DBMS, based on location and importance value [12]. The *Binary Line Generalization-tree* (BLG-tree) [15] takes care of the (line) simplification part of the generalization process.

Further research is required to determine how the GAP-tree can be maintained efficiently under edit operations, instead of rebuilding the complete GAP-tree. A solution for this may be the divide-and-conquer method for large data sets as sug-

[1] We considered explicitly storing the nodes and only storing the intermediate points of the edges. It was decided not to implement this model, because non-topology based visualization of the edges is not possible. The polygon based model was not implemented, because the edges have their own attributes as they are also regarded as features and not only as parts of face boundaries

gested in the previous section. Finally, research is also required to incorporate other generalization techniques to support: symbolization, and displacement [10].

The GAP-tree is very suited for an internet/web vector map application. Initially, only the higher levels of the GAP-tree need to be transmitted for the overview map. This is then filled with more detail until the user stops this process or it is stopped automatically. When a user zooms in, lower level nodes of the GAP-tree may be transmitted, providing more detail on top of the buffer with top level nodes already present at the client side.

Acknowledgments

We would like to thank Harry Uitermark and Frank van Wijngaarden for participating in the discussions during this research. We would also like to thank the two anonymous SDH'98 reviewers for their constructive reviews.

References

[1] Gerard Boudriault. Topology in the TIGER file. In *Auto-Carto 8*, pages 258–269, 1987.

[2] DGIWG. DIGEST – digital geographic information – exchange standards – edition 1.1. Technical report, Defence Mapping Agency, USA, Digital Geographic Information Working Group, October 1992.

[3] Andrew U. Frank and Renato Barrera. The Field-tree: A data structure for Geographic Information System. In *Symposium on the Design and Implementation of Large Spatial Databases, Santa Barbara, California*, pages 29–44, Berlin, July 1989. Springer-Verlag.

[4] J. P. Lagrange, A. Ruas, and L. Bender. Survey on generalization. Technical report, IGN, April 1993.

[5] Martien Molenaar. Single valued vector maps: A concept in Geographic Information Systems. *Geo-Informationssysteme*, 2(1):18–26, 1989.

[6] J. C. Muller, R. Weibel, J. P. Lagrange, and F. Salge. Generalization: state of the art and issues. Technical report, European Science Foundation, GISDATA Task Force on Generalization, 1993.

[7] Thomas K. Peucker and Nicholas Chrisman. Cartographic data structures. *American Cartographer*, 2(1):55–69, 1975.

[8] M. Salzmann, A. Hoekstra, and T. Schut. Quality issues in cadastral map renovation. In *JEC-GI'97, Joint European Conference and Exhibition on Geographical Information, Vienna, Austria*, April 1997. paper presented at Workshop Quality Assurance in Large-Scale Mapping.

[9] Hanan Samet. *The Design and Analysis of Spatial Data Structures*. Addison-Wesley, Reading, Mass., 1989.

[10] K. Stuart Shea and Robert B. McMaster. Cartographic generalization in a digital environment: When and how to generalize. In *Auto-Carto 9*, pages 56–67, April 1989.

[11] Peter van Oosterom. A reactive data structure for Geographic Information Systems. In *Auto-Carto 9*, pages 665–674, April 1989.

[12] Peter van Oosterom. The Reactive-Tree: A storage structure for a seamless, scaleless geographic database. In *Auto-Carto 10*, pages 393–407, March 1991.

[13] Peter van Oosterom. *Reactive Data Structures for Geographic Information Systems*. Oxford University Press, Oxford, 1994.

[14] Peter van Oosterom. The gap-tree, an approach to “on-the-fly” map generalization of an area partitioning. In J.C. Müller, J.P. Lagrange, and R. Weibel, editors, *GIS and Generalization, Methodology and Practice*, pages 120–132. Taylor & Francis, 1995.

[15] Peter van Oosterom and Jan van den Bos. An object-oriented approach to the design of Geographic Information Systems. *Computers & Graphics*, 13(4):409–418, 1989.

[16] Judith van Putten. Experiences with the GAP-tree. Master's thesis, Utrecht University, Computer Science Department, INF-SCR-97-30, November 1997.