5 Large vario-scale datasets

In Chapter 3 the focus was on vario-scale data structure description. This was extended in Chapter 4, where generating better content for this structure was investigated. It showed how the structure has been developed and used in practice, and current technical limitations. One of them is processing really massive dataset with records in order of millions which do not fit in the main memory of computer. It is a notorious and challenging problem. This is especially true in the case of map generalization, where the relationships between (adjacent) features in the map must be considered. Therefore, this chapter presents our solution for automated generalization in vario-scale structure based on the idea of subdividing the workload according to a multi-level structure of the space, allowing parallel processing. More specifically: Section 5.1 specifies our goal. Section 5.2 presents related work and other options to handle large datasets. Section 5.3 explains the principles of our method in more detail. In Section 5.4 modifications of the process specific for road network generalization are introduced. Statistics and a test of real dataset with more than 800 thousand objects are given in Section 5.5, followed by conclusions and the future work related to processing large datasets in Section 5.6

Own publications

This chapter is based on the following of my own publications:

- Meijers, M., Šuba, R., and van Oosterom, P. (2015). Parallel creation of vario-scale data structures for large datasets. *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, XL-4/W7:1–9.

§ 5.1 Requirements

Geographic vector datasets are an example of the big data phenomenon. Practitioners using these large geographic datasets of the whole world, a continent or a country, for example, can easily get into difficulties because the sheer size of the data is too big. Functionalities such as storage, analysing, processing or visualisation hit the physical limitations of a computer when treating all data at once. Automated map generalization is not exceptional in this. This is also true for the process of generating vario-scale data structures, see Chapters 3 and 4. Only datasets which can fit into the main memory of the computer can be processed efficiently, limiting the maximum size of datasets to be processed. Therefore, we have developed an automated generalization method for processing large geographic vector dataset of arbitrary size into vario-scale data structures.

The main goal of our research for this chapter is to create a process to be able to produce a vario-scale data structure for a large dataset based on the following requirements (each one supported by its own motivation):

- Input independent – The approach should be able to process different source datasets intended for various map scales.
- Automatically generated partition – The division of the space into smaller pieces should work fully automatically and not require manual intervention.
- Suitable for parallel processing – Besides the fact that datasets hitting the physical limitation of the computer cannot be processed, the approach should be suitable to run in parallel for more practical reasons, such as reduction of processing time.
- The features should be processed just once – Features on the boundaries of a dataset split in pieces will require special care. However, it should be guaranteed that features should not be split in artificial parts, and the feature is processed just once during the whole process, increasing the efficiency and consistency of the result.
- Be suitable for data produced at a range of map scales – A vario-scale structure contains a whole series of simpler maps where features in the map are modified versions of their own predecessor. This requires a hierarchical data structure to record all the successive changes.

The described method in this chapter fulfils the requirements listed above. It fits to one of our goals to produce a vario-scale topographic dataset (for example based on the large scale 1:10,000 map for the whole of The Netherlands) and disseminate this data via a web service interface enabling access at arbitrary map scale and smooth zoom user interactions over the web, see Section 3.6. This dataset's planar area partition (roads, water, terrain) contains approximately 5.2 million area features. To be able to process a massive dataset of this size we have used the concept of a Fieldtree (Frank and Barrera, 1990) which was proposed quite some time ago to be applied to create tGAP (van Putten and van Oosterom, 1998), but has never been implemented nor tested. The method splits the original dataset into smaller pieces, called *fields* (at multiple levels). Starting at the lowest level, these small fields can be handled separately, and processed fields (with less data after generalization) are combined into bigger fields at a higher level. An additional benefit is the possibility to process the fields in parallel on today's multi-core systems, which can significantly speed up the generalization process. The objects located on the boundaries of the fields require special attention, which will be closely described later in the text.

§ 5.2 Other approaches

Automated map generalization of large datasets is a notoriously difficult problem: It is a computationally complex, time demanding and data-intensive problem and it requires high-performance computing, commonly considered as one main approach for handling such data. Using cluster-, grid-, cloud- or super computing, parallel processing is a good way to deal with massive datasets (Sharker and Karimi, 2014). It requires decomposition into independent tasks that can run in parallel. This can be executed in less time, leading to more efficient computation and solving of complex problems,

which map generalization unarguably is. These reasons and the fact that computers with multiple cores are common in these days explains why researchers focus on map generalization of large datasets in parallel.

Many aspects of processing geographic data in parallel have been extensively studied in the past. Domain decomposition for parallel processing of spatial problems has been studied by Ding and Densham (1996), Armstrong and Densham (1992), Zhou et al. (1998) and Meng et al. (2007). Meng et al. (2007), for instance, used the Hilbert space filling curve to achieve a better parallel partitioning. Another focus point has been on the development of frameworks for parallel computing (see *e. g.* Hawick et al. (2003), Wang et al. (2011) and Guan et al. (2012)). The MapReduce programming model (Dean and Ghemawat, 2004) is well known for parallel computation and has only recently been extended with capabilities for various computational geometry problems (Eldawy et al., 2013). It must be noted, that all these works do not focus on the problem of automated map generalization.

Meijers and Ledoux (2013) present an algorithm, termed EdgeCrack, to obtain a topological data structure and perform small geometric corrections of the input by snapping to avoid problems. They show how they extend the algorithm to a Divide-and-Conquer approach using a quadtree based subdivision, which is also suited for parallel processing. They report on successfully obtaining a topological data structure for 5.3 million polygons with their approach, however it does not consider any generalization.

The Netherlands' Kadaster processes the whole of The Netherlands from scale 1:10,000 to 1:50,000 in an automated fashion in parallel (Stoter et al., 2014). They developed their custom solution using ESRI generalization software. To obtain a map for the whole of the country they partitioned the dataset based on the main road network. Their irregular partitioning requires minimal splitting of features and the amount of data is more less distributed equally. This advantage comes with drawbacks, because defining this partitions is difficult and cannot be done fully automatically (*e. g.* near the coast the road network is too sparse to obtain reasonable small dataset parts). Their solution is created and tuned for generalization of maps for a specific target scale and to achieve the best result possible, however this tailor-made solution can not be applied out of the box for different dataset or targeting creation of data for another map scale such as a European land use map.

Thiemann et al. (2011) present the automated derivation of CORINE Land Cover (scale 1:100,000) from the high resolution German authoritative land cover datasets (scale 1:10,000) of the whole area of Germany. First the datasets are split into rectangular and slightly overlapping tiles. These are processed independently and then composed into one result. To preserve consistency of data some redundancy is added to the partitions in the form of overlapping border regions. In these border regions features are processed more than once, leading to redundant (and potentially inconsistent) output. This redundancy is removed in the composition phase. Their generalization uses some deterministic algorithms for every tile to guarantee that the tiles can be stitched together without any observed problems later. A user parameter that describes the size of the overlapping border regions is required.

```
NRLEVELS=8      # Given number of levels to create
FINEGRID=12500  # Grid size (m) at most detailed level
THEORIGX=0      # Origin in x- and y-direction
THEORIGY=0

for (i=0; i<NRLEVELS; i++) do
    SIZE[i] = FINEGRID * 2^i

ORIGX[NRLEVELS-1] = THEORIGX
ORIGY[NRLEVELS-1] = THEORIGY
for (i = NRLEVELS-2; i>=0; i--) do
    ORIGX[i] = ORIGX[i+1] - SIZE[i+1]/4
    ORIGY[i] = ORIGY[i+1] - SIZE[i+1]/4
```

FIGURE 5.1 Pseudo code as given by van Oosterom and Vijlbrief (1996) for determining the layout of a Fieldtree (its grid size and the shifted origin of the grid cells at every level). Note that the number of levels (needed to be able to fit the whole domain of the dataset in one last field) can be determined given the parameter for the finest grid size (finegrid) and the extent of the dataset.

§ 5.3 Generating a large vario-scale structure

The whole process of obtaining a large vario-scale data structure is composed of a sequence of steps: Data import, constructing a Fieldtree and distribution of the objects over the Fieldtree, processing field by field (level by level) and finalization. The description of the complete process will now follow in the same order in sections, step by step.

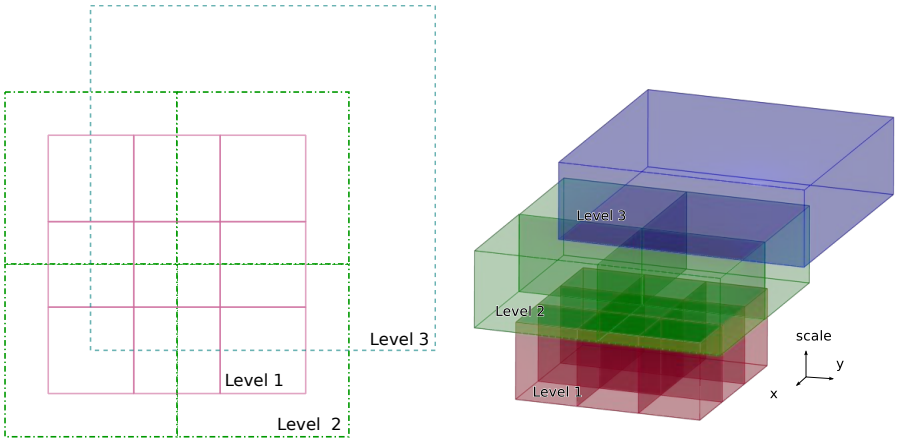
§ 5.3.1 Import

The initial step of the process is to import the planar partition into a topological data structure consisting of nodes, edges and faces. We consider topological clean data where nodes with a degree of two do not exist (unless for island rings) and there are no intersecting or dangling edges. If the input data is not in the required topological structure (as data is often modelled as a set of simple features) and the dataset is large, then this is also a large data processing challenge, to which the same parallel processing strategy could have been applied as presented in this paper.

§ 5.3.2 Constructing a Fieldtree and object distribution

To split the dataset into smaller pieces we use the Fieldtree data structure (Frank and Barrera, 1990). The Fieldtree has been designed for GIS and similar applications. Besides its multi-level organization of space, it subdivides space regularly, spatial objects are never fragmented, and geometric and semantic information can be used to assign the location of a particular object in the Fieldtree.

Figure 5.2 demonstrates the organization of fields at multiple levels in the Fieldtree. The most detailed and smallest fields are at the bottom (large scale), and the less detailed and larger fields (small scale) are at the top of the Fieldtree. The bottom fields will initially be filled with the input data, while the content of the higher level fields is



(a) Three levels where lowest is in pink, middle in green and top level in blue.

(b) 3D impression of the same fields. Note, the third dimension is scale.

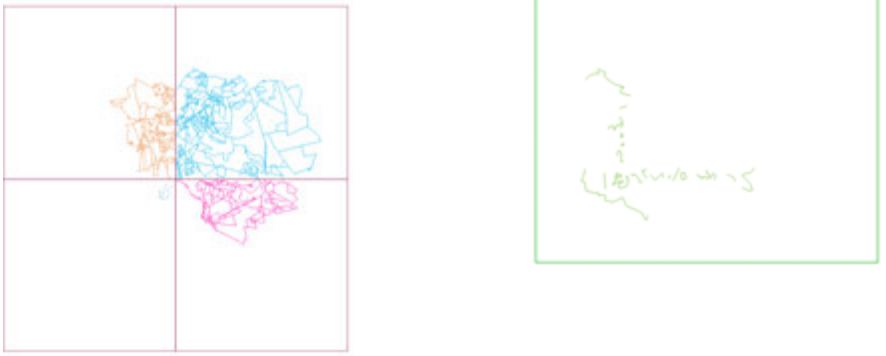
FIGURE 5.2 A Fieldtree (top view and 3D impression)

(mainly) generated during the generalization process. Note that due to the shifted grid origins, every field has nine child fields (of which 1 completely contained, 4 half contained, and 4 quarter contained) in the level below, except the fields from the lowest level. In the reverse direction: a field has either 1, 2 or 4 parents (depending on its location in the grid). The fact that a field can have more than one parent causes it to be a non-hierarchy and strictly speaking the term 'tree' is incorrect. The number of levels in the Fieldtree is based on the extent of the dataset, and the size of the lowest level field is given by a user defined parameter (*cf.* Figure 5.1).

When the layout of the fields for all levels in the tree has been determined the features of the dataset are redistributed over the fields. The nodes, edges and faces are distributed to the fields in the lowest level based on their bounding box, see Figure 5.3. Every feature is stored in the smallest field in which it is completely contained. If the object is bigger than a field or it intersects with more fields, a non-fitting object is placed in a field at a higher level and it will be processed later (together with the generalized features of the child fields). It has been proven that an object is completely contained in a field not more than two levels higher, compared to its own size. A very large object will be placed in one of the highest levels of the Fieldtree, which means that such an object will not be processed (generalized) until that level is reached.

Once the distribution is finished, the lowest level fields can be treated separately in parallel. Every field has it own set of tables with nodes, edges and faces in the database: When a field is processed these tables provide the initial input; during processing these are populated with the result of the generalization steps and at the end of processing output is written to the appropriate parent field tables.

Note that only features completely present in a field are processed at the current level (*i. e.* for an area feature (face) all its composing nodes and edges are present in the field). Features on the boundary of a field are processed at a higher level, where they completely fit. When all the nine child fields are processed, then the parent field one



(a)Lowest level fields (b)Highest field

FIGURE 5.3 Distribution of the edges over the fields of the Fieldtree. Data that fits in a field has its own colour. Edges that intersect the boundary of one of the fields at the lowest level, have been stored in the field at the next level. Note the shift in origin of the field at the highest level.

lever higher can be processed. This processing of fields is repeated until the top level is completed.

The grid location at the next level is shifted, in order to guarantee that objects which could not be processed in the current field (*e. g.* they are bigger than the field) can be treated in the next or one level above that. Note that every field of the next level has edges of twice the size (and therefore four times the area size) compared to the fields at the current level. It is our goal to simplify the content of a field by a factor 4, that is, if the input consists of n area features, then the output should be of size $n/4$. In a uniform distribution this will result in higher level fields with similar workload as in the lower level fields. In case of non-uniform data the reduction by a factor of 4 will maintain the relative differences in map density, while decreasing the number of features towards the higher (and larger) level fields.

§ 5.3.3 Processing Field by Field

When the features have been distributed over the fields of the Fieldtree every field in the lowest level can be processed. Processing fields (*i. e.* executing the automated generalization process for this part of the map and keeping track of the result in a tGAP structure) is the most time and computational consuming part. But because objects in the different fields are not in interaction with each other, every field can be processed separately and this processing can happen in parallel. Note that our automated generalization process is based on iteratively finding the least important feature in the current field, instead of a global criterion, this has now become a 'local field' criterion. This makes sense as generalization of for example the northern part of the dataset, should not influence much the generalization that happens in the south. Furthermore, replacing the global by a local criterion makes the problem computationally easier.

Processing one field means applying generalization operations: Merging faces, removing boundaries (edges), simplification of edges, collapsing areas to line features and splitting of faces. This leads to simplification of the objects of the part of the map stored in the field and divides generalized features into two categories. First, finished features: these are not valid any more for the map scale that has been reached with the generalization process. These features are stored for the final vario-scale dataset. Second, unfinished features: these still need to be processed (together with the 'boundary' features, which did not fit in the lower level fields and the features that were intersecting the field boundaries of the just processed level). These features are placed in the tables of the grid cell of the next level in the Fieldtree in which they completely fit. We call this *propagating* features up. As explained above, to preserve the same amount of information (processing work) through the whole Fieldtree about 75% of the field is processed. The other 25% is left and propagated up. Because the area of a field at the next level is four times bigger, the data amount (and thus the workload) for a field at the next level will be approximately the same.

§ 5.3.4 Finalization

After all fields at all levels of the Fieldtree are processed the final operation takes place; it combines all information together. All processed fields are searched and the final global ordering (and numbering) of the objects based on their range of scales defined during the vario-scale structure creation takes place. This global ordering is helpful when using the structure for making maps at arbitrary map scale. The individual field tables are merged together to create one set of tables (with one node, edge and face table for the final dataset) that encodes the result of the progressive generalization process for the whole dataset. At this stage the structure is ready to use by other applications such as web clients supporting smooth zoom of vario-scale data.

§ 5.4 Read-only buffer zone for road network

When we tried to use our Fieldtree approach for a road network dataset of The Netherlands (in Dutch: Nationaal Wegen Bestand), we realised that for generalization nearby features are important (*e. g.* in the connectivity analysis needed in road network generalization), and this was slightly problematic with our design where features overlapping tile boundaries are already propagated up (not present at lower levels).

We outlined in Section 5.3 that objects on the boundary of a field can not use information about their direct neighbours for generalization decisions, because they are not present (having already been propagated up). However, in case of processing the road network, we need extra information at the boundary of the field because the connectivity of the roads is crucial, because roads incident with many other roads should stay in the process the longest. Therefore we introduced a 'read-only' data buffer around the boundary of the field where no generalization actions can be performed and objects are used only for connectivity information purpose, see the red features in Figure 5.4. The buffer is computed locally. Instead of using a geometric radius ϵ, we use a topological measure, only considering i-neighbours, see Section 4.3.3. It is similar to a breadth-first search, *e. g.* 1-neighbours check only direct neighbours, 2-neighbours check the neighbours of the direct neighbours, etc. In our case, first the faces intersecting with

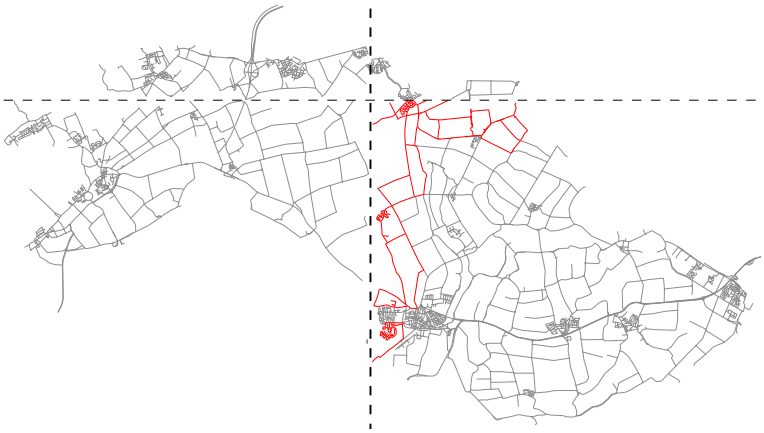


FIGURE 5.4 The inward 'read-only' buffer example for a subset of the road network dataset with only four field at the bottom. The buffer for right bottom field in the figure is illustrated in red. $2 - neighbours$ of the intersecting face with boundary of field has been chosen. Note two things: First, only the content of the fields at current level are shown. Faces intersecting with boundary are not in the figure (already propagated up). Second, the part of the dense region near to the boundary is not selected as a buffer because no intersecting face is within the $i - neighbours$ distance.

the field boundary are selected. Then the 'read-only' buffer based on the i-neighbours of these faces is computed. Since the boundary intersecting faces and buffer faces are propagated up and generalization is only performed in the non-buffer regions less work is done. As a result all generalization happens in smaller region – in the area that is shrunk inwards. When the field is processed, all objects from the 'read-only' buffer are propagated up.

§ 5.5 Statistics

We have tested our method with multiple datasets that differ in content, size and type of features. The details of these datasets (Figures 5.4 and 5.5 give an illustration) are as follows:

- CORINE Land Cover dataset for an area of the United Kingdom and Ireland with approximately 100,000 faces and an area around Estonia with approximately 130,000 faces both intended to be used at a 1:100,000 map scale (Figure 5.5a).
- A road network dataset of the Netherlands (in Dutch: Nationaal Wegen Bestand). This dataset is intended for use at a 1:25,000 map scale. Note that the cycles/areas of the road network (*i. e.* the 'space between the roads') have been used for creating a planar area partition and driving the generalization procedure (iteratively picking the smallest cycle of the road network to be removed, ultimately leading to removal of an road from the network, as described by Šuba et al. (2014b)). It contains approximately 200,000 of these areas.
- Dutch cadastral dataset (scale 1:1,000) for area of the province of Gelderland (The Netherlands) with approximately 880,000 faces. The geographic extent

(a) (b)

FIGURE 5.5 Two of the testing datasets: (a) CORINE and (b) cadastral map

of the data spans 130 km × 90 km. Although the generalization process is less meaningful for cadastral parcels, this dataset is a useful test dataset for our research as it is topologically structured and has been proven to be topologically clean and valid (Figure 5.5b).

All datasets have been processed in parallel (multiple fields at same time) at our research server with 16 CPU cores[1]. All vario-scale tables are stored in a PostgreSQL database management system extended with PostGIS[2].

The different datasets we processed show that the approach works, even for different types of input data (land cover data, road network and cadastral map). The successful processing with the road network dataset demonstrates that when it is important to have neighbouring features available (for generalization decisions), it is possible to shrink the domain inwards. With this approach we are still able to process fields in parallel, while each feature is just processed once.

To obtain insights into what is a reasonable field size for the smallest fields in the fieldtree, we ran the whole process (constructing fieldtree, distributing the objects, generalization field by field, finalization) with different sizes for the smallest fields in the fieldtree. We varied the sizes of these fields, based on the average size of a feature in the dataset. We used sizes varying from 20 to 160 times the average feature size. Figure 5.6 shows the resulting timings we obtained. The graph shows that as rule of thumb, we can set the size of smallest fields to 100 times the average feature size (*i. e.* around 10,000 objects in the smallest fields) and that the total runtime is then rather

1 HP DL380p Gen8 server (with two 8-core Intel Xeon processors, E5-2690 at 2.9 GHz, 128 GB main memory, and RHEL 6 operating system) with Disk storage (direct attached) with 400 GB SSD, 5 TB SAS 15K rpm in RAID 5 configuration (internal), and 2 × 41 TB SATA 7200 rpm in RAID 5 configuration.

2 PostgreSQL 9.3.4 and PostGIS 2.2.0dev

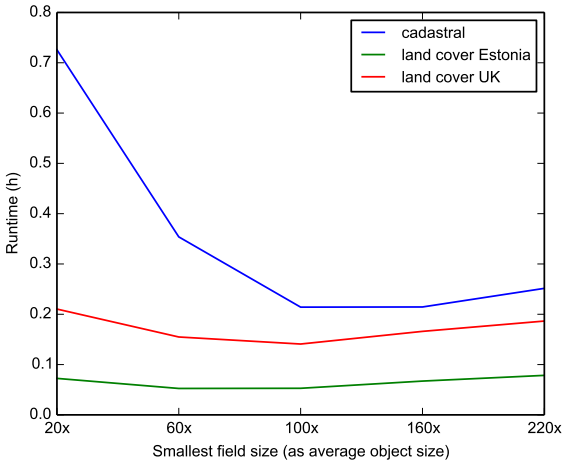


FIGURE 5.6 Processing time needed versus field size (smallest fields). Note that the smallest field size was determined as a multiple of the average feature size in the dataset.

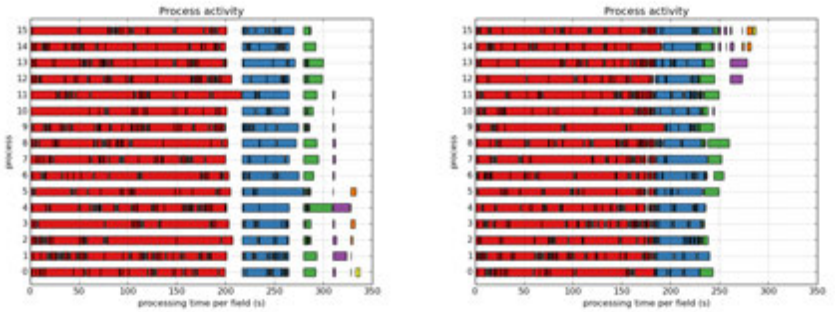
optimal. Furthermore, this graph teaches us that too small fields lead to a large overhead in processing: Many tables that contain a little bit of data cannot be generalized sufficiently for the next level (so no reduction will take place at a level, wasting processing time / copying data to the next level).

For the biggest dataset (the cadastral map with 880,000 faces) we investigated in-depth the runtime needed for the generalization step. In a sequential run this step took 7900 seconds (slightly more than 2 hours). We measured the time needed for parallel processing field by field. We also investigated whether this could be shortened by taking a different order of processing the fields. Figure 5.7 shows two possible scheduling strategies for distributing work in multiple parallel CPUs.

The first strategy, the 'level order' scheduling, takes care of all fields at the same level before the higher level is started. The second and more refined strategy, the 'parent child' scheduling, will schedule the parent field for scheduling if all its children are processed.

We assumed that the second scheduling option would significantly reduce the processing time, but this was a bit disappointing. A modest reduction was observed – from 340 to 290 seconds (15%). The main bottleneck remains near the top of the Fieldtree where most of the CPUs have to wait while only few processors are used. Note that the processing time of a field is dependent on the amount of data in that field. This might be used in a better scheduling strategy.

We found having processed the CORINE land cover dataset that the majority of the generalization work was performed towards the end of the whole process (40% of the runtime was devoted to processing the last field). After inspection, this turned out to be caused by some very large polygons remaining to the end of the process (these massive polygons with a lot of holes representing the land between all small settlements, which



(a)Level order scheduling (b)Parent child scheduling

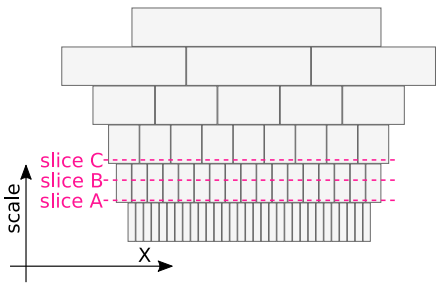
FIGURE 5.7 Two different scheduling orders for the cadastral dataset. Every rectangle represents one field. Note that different colours correspond to the levels of the fieldtree (*i. e.* red corresponds to the fields with finest grid size). (a) Level order scheduling: The process waits until all fields of the same level are processed before processing of the next level starts. (b) The parent child scheduling: Fields are scheduled for processing when all its up to 9 child fields are done.

only fitted in the largest field of the Fieldtree). Apart from having a negative influence on the run time, these polygons will also have a negative impact on the cartographic result as these polygons will keep their full original amount of detail until when they are processed.

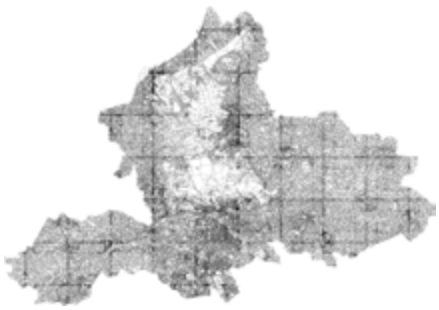
In order to assess the visual impact, we performed two following tests:

- We compared the result of the parallel version of the algorithm with the sequential run (original non-parallel approach); see Figure 5.8. The visual impression of both approaches are quite similar. The main difference is that the number of vertices is higher in the parallel version (counting showed that this is roughly twice the number of vertices). We assume this is due to number of lines fitting in the field, *i. e.* even longer lines fit in the field in the sequential run, hence more line processing could be performed.
- We selected three output maps at specific locations, in order to observe the effect of the regular grid on cartographic quality of the end result, see Figure 5.9. We chose three locations; *Slice A*, right after the lowest field level, *slice B*, in the middle of the second field level, *slice C* at the beginning of the third. The most visible 'seams' are right after previous level because almost no generalization took place on the field boundary compared to the generalized centre of the field. Note that user's viewport is smaller than any of the field of the grid. This means user would not have enough context information on the display to observe differences between generalized and less generalized regions.

FIGURE 5.8 The visual comparison of a sequential run (in left column) and run in parallel (in right). The rows display different feature counts: 250, 500 and 1000 from top to bottom.



(a)Fieldtree – Side view for the cadastral dataset. Dashed lines indicates location of output maps captured in 5.9b, 5.9c and 5.9d.



(b)Slice A

(c)Slice B

(d)Slice C

FIGURE 5.9 The effect of regular subdivision of space on cartographic quality. An example of the cadastre dataset shows that 'seams' may be the most visible when inappropriate location in the Fieldtree is selected such as the beginning (b) or the end (d) of field level.

§ 5.6 Conclusions

An approach for automated map generalization to obtain a vario-scale data structure of massive datasets has been proposed and tested. We have described all steps of the process chain and we tested the approach with real data. It has been shown that our approach is generic and works for different types of input data such as land cover or road network data. We have tested with real world data , although we still plan to test the approach with a really massive dataset (*e. g.* topographic dataset of The Netherlands or CORINE for whole of Europe) which will not fit in main memory. We expect that larger dataset will benefit from having even more parallelism assuming the hardware is available, *e. g.* 32 processors. The design is scalable: each job gets its input data from the database and writes the output again to the database. So, the whole dataset does not have to fit in main memory. As long as the active jobs do fit in memory, the process should remain efficient. We have to explore this further. If we keep the same amount of main memory but increase the number of parallel processes, will these jobs still fit in main memory? If not, we should either make the average job size smaller or add more memory.

The Fieldtree is very appropriate as it is good start for divide-and-conquer, but also offers very useful multiple levels: What is not solved at the lowest level can be solved at higher (which glues partial solutions from lower level fields together). This fits very well with our approach to map generalization, but the divide-and-conquer approach using the Fieldtree may not be limited to map generalization only, and other spatial problems could benefit from this type of organization as well, *e. g.* obtaining an explicit topological data structure.

We have presented two approaches for scheduling the parallel work. We are aware of the fact that both of the scheduling approaches are still sub-optimal. An improved scheduling might be based on processing the largest fields (with respect to number of features contained) first and smaller later, and applying dynamic scheduling from a 'work-pool'.

Furthermore, we only carried out limited evaluation of the cartographic end results and as we found out with processing the CORINE land cover dataset with massive polygons, our approach is less robust against these polygons.

This shows that there are still some open questions to be answered. Our future work includes:

- Very large features survive until the end and are not generalized. These large objects and also their island objects are affected. In our tests typical cases are: large 'background' polygons, or very long infrastructure features, such as rivers and roads. Our proposal for a solution in this case: split the large features. This raises another interesting challenge: How to best split these large features into smaller (artificial) chunks.
- The effect of the regular grid on cartographic quality of the end result can be reduced. The 'seams' of the grid are caused by non-processed features because they lie on boundary of a field. Therefore we propose two options; a) to use two grids at every level (the second one shifted to the middle) and have two gener-

alization steps of 37,5% (instead of one 75% generalization), b) to have smaller steps between levels (and less generalization per level), result is more levels, but also more arbitrary off-sets.

- The proper size of the lowest fields need further investigation. What is a good balance between size of fields, running time and cartographic result? We showed that the running time depends on the field proportions and number and sizes of the features within. Too small fields lead to a lot of overhead in processing. Too big fields can lead to memory problems. This implies that a proper size of fields is crucial for obtaining a good cartographic end result. The size of fields is reflected in generalization quality. If the fields are big the 'seams' of the field may be visible, because the objects on the boundary could not be generalised for a long time while the rest of the field has been simplified a lot. By contrast too small fields will contain fewer objects which leads to fewer generalization options again leading to a worse result.
- The Fieldtree subdivides space regularly where the density of the original data is not considered. In some cases a large city is in one field (lot of generalization processing is needed) while the rest of the fields cover the rural regions where not much simplification is required. Then most of the resources are spent on the field containing the city while the other fields are already processed. A good strategy would be to locally deepen the fields at lowest level where high data density regions are reached. This would result in a slightly different Fieldtree: a structure of which the lowest deepened fields are not covering the whole domain any more
- An influence of the parallel creation compared to the traditional creation (based on global criterion for least important feature) on cartographic quality is not verified. For example is it possible to distinguish area processed by a different field? Does the user perception change when we use different/modified grid of the Fieldtree? Based on initial visual inspections we assume this is probably not the case, but should be verified with user tests.
- As the parallel creation is not based any more of the global criterion (of least important feature), the Fieldtree structure can be also used for the update of vario-scale structures: only the field with updated data needs to be reprocessed, and only if generalized result of field is significantly different, then also the parent(s) needs to be reprocessed (recursively).