RESEARCH ARTICLE

Area aggregation in map generalisation by mixed-integer programming

JAN-HENRIK HAUNERT[†*] and ALEXANDER WOLFF[†]

[†]Chair of Computer Science I, University of Würzburg, Am Hubland, D-97074 Würzburg

Topographic databases normally contain areas of different land cover classes, commonly defining a planar partition, that is, gaps and overlaps are not allowed. When reducing the scale of such a database, some areas become too small for representation and need to be aggregated. This unintentionally but unavoidably results in changes of classes. In this article we present an optimisation method for the aggregation problem. This method aims to minimise changes of classes and to create compact shapes, subject to hard constraints ensuring aggregates of sufficient size for the target scale. To quantify class changes we apply a semantic distance measure. We give a graph theoretical problem formulation and prove that the problem is NP-hard, meaning that we cannot hope to find an efficient algorithm. Instead, we present a solution by mixed-integer programming that can be used to optimally solve small instances with existing optimisation software. In order to process large datasets, we introduce specialised heuristics that allow certain variables to be eliminated in advance and a problem instance to be decomposed into independent sub-instances. We tested our method for a dataset of the official German topographic database ATKIS with input scale 1:50,000 and output scale 1:250,000. For small instances, we compare results of this approach with optimal solutions that were obtained without heuristics. We compare results for large instances with those of an existing iterative algorithm and an alternative optimisation approach by simulated annealing. These tests allow us to conclude that, with the defined heuristics, our optimisation method yields high-quality results for large datasets in modest time.

Keywords: map generalisation, aggregation, combinatorial optimisation, mixed-integer programming, NP-hardness

1. Introduction

In recent years, researchers have made considerable advances in quantifying the quality of map generalisation (Bard 2004, Cheng and Li 2006, Frank and Ester 2006). Usually, measures have been proposed for assessing the outcome of given generalisation procedures. For example, Cheng and Li (2006) discuss quality measures for polygon maps, that is, partitions of the plane into polygonal regions of different classes. The authors performed experiments with different settings of a simple generalisation method: areas that are too small for the target scale are merged with neighbours, which potentially have different classes. They compared the results with respect to the area that changes its class in generalisation, which they define as a global measure of "semantic consistency". In this article, we define a similar quality measure based on class changes. We, however, do not only apply this measure to assess generalisation algorithms, but also present an aggregation method that yields results of maximum quality under given hard constraints. This method is based on mixed-integer programming, which is a technique for combinatorial optimisation. Though we need to introduce heuristics to process large

*Corresponding author. Email: jan.haunert@uni-wuerzburg.de

datasets (the problem is NP-hard as we prove in Section 4), our method yields the exact optimum for small samples. This offers new possibilities to assess the result of heuristic generalisation methods.

Usually, two different generalisation problems are distinguished (Brassel and Weibel 1988): the derivation of a less detailed database from a given one (database generalisation or model generalisation) and the derivation of a graphical map, either from a database or another map (cartographic generalisation). In this article we address database generalisation, which aims at data abstraction rather than at graphical effectiveness. As a prerequisite for automatic database generalisation, national mapping agencies in several countries have developed database specifications, often defining minimal dimensions that need to be satisfied in a particular scale (Afflerbach *et al.* 2004). In order to ensure logical consistency, we consider such requirements as hard constraints that define the set of feasible generalisation solutions. We focus on the generalisation of planar partitions since these are commonly used to represent land cover information. This implies that we do not allow gaps or overlapping areas. A common approach to satisfy size constraints for such a database is to *aggregate* areas, that is, to merge several areas into one. Most often this task is solved by iteratively merging pairs of areas, see Section 1.1. The problem has not been approached by optimisation yet.

Clearly, the given hard constraints allow for different feasible solutions. Optimisation means to search for the solution of highest quality among them. In map generalisation, the quality is often expressed by a set of soft constraints, that is, constraints that allow different degrees of satisfaction. Typically, the soft constraints are conflicting and compromises need to be found (Weibel and Dutton 1998). In our approach, we optimise compactness of shapes and semantic accuracy. Our measure for the latter is based on class changes, see Section 1.2.

Existing approaches to combinatorial optimisation problems in map generalisation are mainly based on meta-heuristics that iteratively improve the map, such as hill climbing (Regnauld 2001, Galanda 2003), simulated annealing (Ware *et al.* 2003), or neural networks, which are applied in self-organising maps (Sester 2005). Some approaches are able to organise multiple generalisation operators. For example, the system of Galanda (2003), which is based on a multi-agent paradigm, integrates algorithms for reclassification, aggregation, typification, displacement, exaggeration, collapse, elimination, enlargement, simplification, and smoothing. Though we focus on the aggregation of areas, such an overall strategy is needed. A global strategy, however, can only be successful if the underlying algorithms provide solutions of high quality. Therefore, we also see our work as a contribution to a better solution of the whole generalisation task. To produce well-generalised results, we have also developed a method for area collapse (Haunert and Sester 2008) and implemented a line simplification algorithm similar to that of de Berg *et al.* (1998). We propose to apply these procedures in succession: collapse of narrow polygons (for example, rivers), area aggregation, and line simplification. From now on, we refer to the result of the collapse procedure as input; this is what our aggregation method processes.

Mixed-integer programming has often been applied to spatial allocation problems, for example, sales territory alignment (Zoltners and Sinha 1983) and school redestricting (Caro *et al.* 2004). In many of these problems, size, compactness, and contiguity of the output regions are important criteria. Since these criteria are also important for map generalisation, we can reuse some of the existing mixed-integer programming formulations. On the other hand, some of the innovations that we introduce may be interesting not only in the map generalisation context but also in the general context of spatial allocation problems. In particular, we present a flow-

based MIP that allows us to group a set of minimal mapping units into contiguous and optimally compact districts that satisfy size constraints (see Appendix B). This MIP gets by with a linear number of variables and constraints without requiring a set of predefined district centres. Furthermore, the heuristics that we present may be successful for many spatial allocation problems.

The outline of the article is as follows. We first review existing approaches to aggregation in map generalisation (Section 1.1) and sketch our notion of relevant quality elements (Section 1.2). We then give a quick introduction into the basics of the optimisation technique we apply and the fundamental differences between this technique and the prevailing iterative methods (Section 2). We present a formal problem definition (Section 3), a proof of NP-hardness (Section 4), our mixed-integer programming formulations (Section 5), and heuristics that successfully tackle the high complexity of the problem (Section 6). We present our experimental results in Section 7 and conclude the article in Section 8.

We provide three appendices as supplementary on-line material. Appendix A sketches the basics of mixed-integer programming. Appendix B presents our flow-based MIP that allowed us to solve small problem instances with proof of optimality. Appendix C presents a large sample processed with our heuristics.

1.1. *Aggregation in map generalisation*

According to Timpf (1998) aggregation is the most common hierarchy that exists among objects of different scales: as a part of data abstraction, a group of objects, for example small forest areas, is replaced by a single object (an aggregate of forests) in a smaller scale. Defining such groups is a key task of map generalisation that has been investigated for different types of objects, for example buildings (Boffet and Serra 2001) and islands (Steiniger and Weibel 2007). Grouping is often done according to principles of human perception, which, in early works of psychology, were subsumed by laws of so-called Gestalt theory (Wertheimer 1938). For example, grouping of objects is done according to their proximity. In our approach we consider this criterion by measuring the compactness of shapes, that is, parts of the same aggregate should be close to a common centre.

Several researchers have proposed simple iterative algorithms for the aggregation of areas in a planar partition. A common approach is to iteratively select an area from the dataset and to merge it to one of its neighbours. This neighbour is selected according to a local compatibility measure, for example, a function of class similarities and common boundary lengths (van Oosterom 1995). Algorithm 1 formalises the iterative approach in a rather general way. The algorithm terminates when all areas have sufficient size for the target scale.

Algorithm 1 Iterative aggregation of areas (region growing)

1: $S \leftarrow$ set of areas below threshold for target scale
2: **while** $S \neq \emptyset$ **do**
3: $\quad a \leftarrow$ smallest area in S
4: $\quad$ Merge a to most compatible neighbour.
5: $\quad$ Update S.
6: **end while**

For similar algorithms, Podrenek (2002) and van Smaalen (2003) define criteria that are evaluated in each step, in order to select areas that are to be aggregated. Jaakkola (1997) applies a similar iterative aggregation algorithm within a system for the generalisation of raster-based land cover maps. Also Cheng and Li (2006)

(a) input

(b) result of Algorithm 1, applying specifications for scale 1:250,000

Figure 1.: A sample from the German topographic database ATKIS.

merge areas that are too small with neighbours. They compare two different settings of this method. In the first setting, the neighbour is selected according to its size; in the second setting, it is selected according to the common boundary length. With the first setting, Cheng and Li observe for one dataset that 12.3% of the total map area changes its class (8.3% with the second setting). They refer to this measure as "semantic consistency". Of course, these values depend on the characteristics of the dataset and the defined thresholds for the target scale. In any case, the experiments show that the amount of class change can be considerable.

We processed a dataset from the official German topographic database ATKIS at scale 1:50,000 (DLM 50) with Algorithm 1 to meet the area thresholds from existing specifications for the target scale 1:250,000 (DLM 250). Figure 1 shows the input for the aggregation algorithm and the result that we obtained by selecting the neighbour according to class similarity values. Though the algorithm produces a clearly less cluttered dataset, we observe a certain shortcoming: each of the red settlement areas in the sample is too small for the target scale 1:250,000. However, a human cartographer would probably create a settlement of sufficient size as their aggregate. Since Algorithm 1 is greedy and takes only direct neighbours into account, it is unlikely that it yields such high-quality solutions. As a consequence, the settlement is lost and relatively large parts of the sample change their classes. Examples like this motivate to approach the problem by global optimisation.

1.2. *Logical consistency, completeness, and semantic accuracy*

We consider three different elements of spatial data quality in our approach, namely logical consistency, completeness, and semantic accuracy. An introduction to quality aspects is given by Morrison (1995).

Logical consistency means compliance with database specifications, that is, we need to ensure structural characteristics of the data model and satisfy constraints on features. In our case the generalised dataset must be a planar partition. Specifications usually define selection criteria for areas based on their size, for example, a forest in the ATKIS DLM 250 must not be smaller than 400,000 m^2 (AdV 2003). We see such criteria as hard constraints, which must be satisfied in any case.

Additionally, we can take a selection criterion as an instruction to ensure completeness: if there is a forest of at least 400,000 m^2 in the source map, it must be present in the DLM 250. We basically assume that all areas can change their classes. However, our method allows us to fix the classes of some areas, for example, to ensure that a forest area of more than 400,000 m^2 will not be lost. Though this restriction can be intended, we refer to this technique as a heuristic.

To measure the semantic accuracy we introduce a semantic distance between

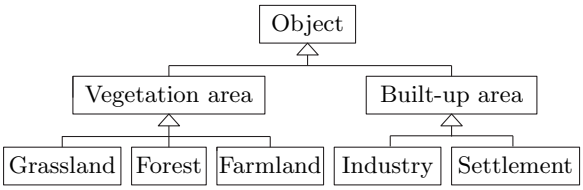


Figure 2.: Class hierarchy defined in the specifications of the German database ATKIS (AdV 2003).

Table 1.: Semantic distance matrix.

	Settlement	Industry	Farmland	Grassland	Forest
Settlement	0	20	100	100	100
Industry	20	0	100	100	100
Farmland	100	100	0	10	30
Grassland	100	100	10	0	30
Forest	100	100	30	30	0

the classes of corresponding areas in the generalised dataset and the input. We formally define the semantic distance as a function $d : \Gamma \times \Gamma \to \ ^{+}_{0}$, with Γ being the set of classes. Small values of d correspond to semantically similar classes. As a global measure we use the average semantic distance (weighted by area)

$$\bar{d} = \frac{\sum_{v \in V} w(v) \cdot d(\gamma(v), \gamma'(v))}{\sum_{v \in V} w(v)}, \tag{1}$$

with V being the set of areas in the input, $w : V \to \ ^{+}$ being their sizes, $\gamma : V \to \Gamma$ their original classes, and $\gamma' : V \to \Gamma$ their classes after generalisation. We do not present a new method to set up the class distances d, but refer to Schwering (2008), who reviews existing approaches. In the map generalisation context, Yaolin *et al.* (2002) propose a method to derive semantic distances and similarity values from given data models. Attribute definitions and class hierarchies as shown in Figure 2 are exploited. For example, since farmland and grassland are both classes of vegetation, they are semantically similar. A reasonable semantic distance measure between two classes a and b, which is used by Yaolin *et al.* (2002), is the minimal number of links from a to a common superclass of a and b. Since this distance is not symmetric, we do not assume that it satisfies the definition of a metric.

Table 1 shows a part of a semantic distance matrix that we generated at our own discretion for classes of the ATKIS database. Our intention was to define a simple setting for experiments. We processed all presented samples with this setting. The distances were mainly set up in a subjective way, but we also considered given class hierarchies. As we aim to minimise the average class distance, the values of d can be seen as costs. For example, changing a unit area from grassland to forest is three times more expensive than changing the same area to farmland.

2. Applied optimisation techniques

Since we apply mathematical programming and heuristics for solving the area aggregation problem, we briefly discuss these combinatorial optimisation approaches.

Mathematical programming is a general approach to formalise and solve a combinatorial optimisation problem. A prominent special case of mathematical programming is *linear* programming. A linear program (LP) consists of two parts: a linear *objective* (or *cost*) *function* and a set of linear *constraints*, each in the same variables. Each constraint corresponds to a half space; the intersection of the half spaces—a (possibly unbounded) convex polytope—represents the set S of feasible solutions. Among the points in S we are interested in one that minimises the objective function. Linear programs can be solved efficiently, for example, using the interior point method of Karmarkar (1984). The simplex algorithm of Dantzig (1963) is efficient in practice but requires exponential time in theory.

In an *integer* linear program (IP) all variables are restricted to integer values.

Usually, the optimal solution of the IP is far from the optimal solution of the corresponding LP. The general integer programming problem is NP-hard. A *mixed-integer* linear program or simply mixed-integer program (MIP) is the generalisation of both an LP and an IP; it can contain both unrestricted (continuous) and integer variables. All our formulations fall into this category.

There are powerful techniques for solving MIPs, and intensive research is being carried out to further improve these techniques. Therefore, formalising an optimisation problem as a MIP is valuable if no efficient, specialised algorithm for the problem is known. With this approach, one will directly benefit from improvements of the general solution techniques. Usually, there are several alternatives to express the same problem as a MIP; choosing among them can result in significant differences in terms of performance. We therefore tested several different formulations of the area aggregation problem. We solved our MIPs with a method called *branch-and-cut*, which we sketch in Appendix A. We refer to Mitchell (2002) for a more detailed discussion. Branch-and-cut techniques are implemented in several commercial software packages. We used the software ILOG CPLEX™ 11.2.

Heuristics do not offer a globally optimal solution, but often yield relatively good solutions in reasonable time. We distinguish heuristics that are designed for a specific problem and those that offer solutions for a very general class of problems *(meta-heuristics)*. We will introduce heuristics of the first type in Sections 6.1–6.2 to eliminate some variables in our mixed-integer programs and to decompose a problem instance into smaller instances. Prominent meta-heuristics are *hill climbing* and *simulated annealing*. Both have been applied to map generalisation problems (Ware and Jones 1998). Starting from an initial solution, a hill-climbing algorithm iteratively moves to a better solution in a defined neighbourhood until no such solution can be found. Especially due to its efficiency and capability of handling multiple operators, hill climbing has been applied to map generalisation (Galanda 2003, Regnauld 2001). However, the hill-climbing approach only leads to local optima. Simulated annealing is based on an algorithm similar to hill climbing, but, in order to escape local optima, solutions of lower quality are occasionally accepted (Kirkpatrick *et al.* 1983). At each iteration, a neighbour of the current solution is randomly selected and accepted with a certain probability, depending on its cost and a predefined annealing schedule. This approach has been used by Ware *et al.* (2003) to comprehensively perform displacement, size exaggeration, deletion and size reduction for multiple map objects. A general problem of simulated annealing is the tuning of parameters that are not inherent to the problem. For example, a cartographer is usually able to specify parameters expressing his preferences for generalisation, but setting up an annealing schedule requires a considerable amount of experimenting. Nevertheless, we use simulated annealing as benchmark to measure the quality of our specialised heuristics. We sketch our simulated annealing set-up in Section 6.3.

In contrast to heuristics, the use of mathematical programming gives us access to exact solution methods, such as branch-and-cut. Though we cannot directly apply the exact methods under time constraints (we will need to introduce heuristics to break down the complexity of the NP-hard problem), we see several advantages compared to iterative meta-heuristics:

+ Optimally solving small problem instances allows the defined constraints and optimisation objectives to be verified independently from heuristics and tuning parameters of algorithms.
+ Iterative meta-heuristics often require hard constraints to be relaxed, in order to produce solutions sufficiently close to the optimum (Michalewicz and Fogel 2004). In Section 6.3 we discuss this problem with respect to area aggregation.

+ The approach by mathematical programming is deterministic, randomised algorithms like simulated annealing are not. Using a pseudo-random number generator, every time with the same initialisation, eases this problem. For other researchers, however, a reproduction of experimental results is still difficult.

On the other hand, mathematical programming has two severe limitations:

− The possibility to express hard and soft constraints is limited, often restricted to linear expressions.
− The time required for finding a (close-to-) optimal solution is difficult to predict.

We will see in Section 3.2 that the first limitation especially affects the possibility to express the compactness of a shape.

3. Modelling area aggregation as optimisation problem

In this section we first give a basic definition of the aggregation problem (Section 3.1) and then discuss possibilities to measure the compactness of shapes. Researchers have proposed many different compactness measures for the analysis of shapes. MacEachren (1985) gives a detailed discussion. We investigate one of these classical measures in Section 3.2 and then explain an alternative approach, which is often applied in mixed-integer programs for districting problems (Section 3.3).

3.1. *Problem statement*

Figure 3 shows an instance of the aggregation problem and a solution. We adopt the definition of the semantic distance d from Section 1.2, as well as the definitions of V, w, Γ, γ, and γ'. However, we use the terminology of graph theory in this section, that is, we refer to nodes instead of areas, weights instead of area sizes, and colours instead of land cover classes.

Problem (AreaAggregation): Given

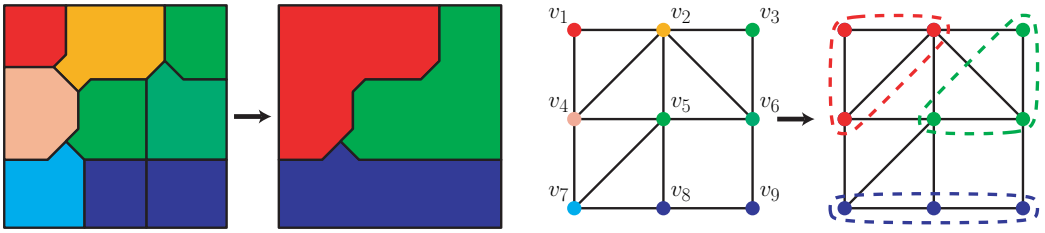
(i) a planar graph $G(V, E)$ with node weights $w : V \to \ ^+$ and a node colouring $\gamma : V \to \Gamma$, where Γ is the set of all colours,
(ii) a function $\theta : \Gamma \to \ ^+$ (a minimally allowed weight for each colour),
(iii) a function $d : \Gamma \times \Gamma \to \ ^+_0$ (the semantic distance),
(iv) a function $c : 2^V \times \Gamma \to \ ^+_0$ (the non-compactness of an aggregate), and
(v) a scalar weight factor $s \in [0, 1]$,

define a new colouring $\gamma' : V \to \Gamma$ of nodes and find a partition $P = \{V_1, V_2, \ldots, V_p\}$ of V such that

(vi) for each node set $V_i \in P$
- all nodes in V_i get the same new colour $\gamma'_i \in \Gamma$, i.e., $\gamma'(v) = \gamma'_i$ for all $v \in V_i$,
- V_i has total weight at least $\theta(\gamma'_i)$,
- the graph induced by V_i is connected, and
- at least one node $v \in V_i$ keeps its old colour, that is, $\gamma'(v) = \gamma(v)$, and

(vii) the cost $f = s \cdot f_{\text{recolour}} + (1 - s) \cdot f_{\text{non-compact}}$ is minimised, where
- $f_{\text{recolour}} = \sum_{v \in V} w(v) \cdot d(\gamma(v), \gamma'(v))$ and
- $f_{\text{non-compact}} = \sum_{V_i \in P} c(V_i, \gamma'_i)$.

The graph $G(V, E)$ in (i) is the adjacency graph of the areas in the input dataset. The node set V contains an element for each area. The set E contains an edge between two nodes if the boundaries of the corresponding areas share at least one line segment that is not degenerated to a point.



(a) map representation (b) graph representation

Figure 3.: An instance of the aggregation problem and a solution, both displayed in map and graph representation. The solution corresponds to the partition $P = \{\{v_1, v_2, v_4\}, \{v_3, v_5, v_6\}, \{v_7, v_8, v_9\}\}$. Assuming that all areas in the input have unit size, all aggregates satisfy the weight threshold $\theta = 3$.

Together, the expressions in (iii)–(v) define the ingredients of the cost function (vii), that is, the generalisation objective. The objective is subject to a number of *hard* constraints, that is, constraints that any feasible solution must fulfil. We now motivate the hard constraints, which are listed in (vi).

Each element V_i in the partition P defines an aggregate, that is, an area in the target scale. Its shape is the union of the shapes that correspond to the nodes in V_i and its class is defined by γ_i'. Each aggregate must be *unicoloured*, *weight-feasible*, and *contiguous*. An aggregate is unicoloured if all nodes that belong to it receive the same colour γ_i'. To be weight feasible, the size of the aggregate must be at least $\theta(\gamma_i')$. This allows us to define different size thresholds for different classes, which, for example, the ATKIS specifications (AdV 2003) do. The requirement for contiguity means that we forbid multi-part features. In this context, the terms connectivity and contiguity refer to the same concept. We use the term connectivity when referring to the graph representation. Additionally, we require that each aggregate contains a node of unchanged colour to avoid that new classes appear in the generalised map. Throughout this article such a node, which defines the colour of an aggregate, will be referred to as *centre*. Note that according to this definition each node is a potential centre.

The cost function (vii) expresses the cartographer's preferences for different feasible solutions. Two different sub-objectives can be identified. The first aim is to change the original classes as little as possible. The second aim is to make shapes as compact as possible. To model these two aims, the functions d and c, which are introduced in (iii) and (iv), are combined in a weighted sum defined by the factor s (see (vii)). The class distance function d is defined according to Section 1.2. It is explicitly given by a quadratic matrix with $|\Gamma| \times |\Gamma|$ elements. Minimising f_{recolour} implies to minimise $\bar{d}$, as the denominator in equation (1) is constant for a given dataset. The function c defines a penalty for the non-compactness of an aggregate, that is, an area in the target scale that is defined by a subset of nodes and their new colour. We assume that c attains high values for complex shapes, but we will simply use the term compactness measure in the following.

3.2. *Compactness based on the perimeter*

The first compactness measure investigated by MacEachren (1985) is $c_1 = \text{perimeter}/(2 \cdot \sqrt{\pi \cdot \text{area}})$. This results in $c_1 = 1$ for circles and in higher values for less compact shapes. Another important feature of this measure is its size invariance, that is, the value of c_1 does not change if a shape is scaled.

We now discuss the setting $c := c_1$ for our global optimisation problem, that is, for each aggregate, we charge an individual cost equal to c_1. Assume that we have two different solutions for the same instance, both containing aggregates that only

differ in size, for example, one solution contains four square-shaped aggregates of size 2 m×2 m, and the other solution contains 16 squares of size 1 m×1 m. If we now charge the same cost for the non-compactness of each aggregate, the first solution with few large aggregates will be preferred, simply because a smaller number of equal penalties is charged. To define a measure that results in equal costs for the discussed partitions, we can charge an individual cost proportional to the size of an aggregate, that is, $c_2 = \text{area} \cdot c_1 = \sqrt{\text{area}} \cdot \text{perimeter}/(2\sqrt{\pi})$. Indeed, setting $c := c_2$ seems to be a good choice, as the shapes in both solutions do not differ in terms of compactness. The formula for c_2, however, contains a square root of a variable measure, that is, the area of an aggregate in the output. Unfortunately, we cannot express this using linear terms only.

At this point, our problem clearly exceeds the possibilities of mixed-integer linear programming. Note, however, that modelling a problem always requires a trade-off to be found between the adequacy of the optimisation objective and the possibility to solve the problem. With this in mind, let's assume that the resulting aggregates in the output have a reasonable and constant size. In this case we can simply define

$$c_{\text{perimeter}} = \text{perimeter}, \tag{2}$$

and set $c := c_{\text{perimeter}}$, which indeed can be expressed in a MIP (Wright *et al.* 1983). Certainly, our assumption is not very realistic, that is, by minimising the perimeter of aggregates, our method will again prefer results with few large aggregates. To cope with this, we could introduce additional requirements, for example, we could define an upper bound for the size of aggregates or a lower bound for the number of elements in P. In Section 6.1 we will introduce a heuristic approach that is based on a set of nodes that are predefined as centres. With this, we can also avoid too large aggregates, if we define that an aggregate must not contain more than one centre. Our assumption becomes much more realistic with this requirement, since an aggregate cannot become too small due to the hard threshold constraints and not too large if we appropriately define the centres. Results of this approach are presented in Section 7. In the following, we will formally refer to $c_{\text{perimeter}}$ as a function of a set $V' \subseteq V$.

3.3. *Compactness based on distances to a reference point*

Let's now discuss another approach to measuring the compactness of an aggregate, which, in similar versions, has often been applied in mixed-integer programs for districting problems (Hess and Samuels (1971); Cloonan (1972); Zoltners and Sinha (1983)). For many applications, these measures have a very concrete meaning: often it is aimed to minimise travel distances of customers to central facilities like stores. This is done by the discussed measures.

Let $\delta : V \times V \to \mathbb{R}_0^+$ be the Euclidean distance between the centroids of two areas. For $V' \subseteq V$ and $\gamma' \in \Gamma$ we define

$$c_{\text{distance}}(V', \gamma') = \min \left\{ \sum_{v \in V'} w(v) \cdot \delta(v, u) \;\middle|\; u \in V' \wedge \gamma(u) = \gamma' \right\}, \tag{3}$$

that is, we select one node $u \in V'$ of colour γ' as a reference point and, for each node $v \in V'$, charge a cost equal to the product of the weight of v and its distance to u. The reference point is selected such that the total cost for an aggregate is minimal. Figure 4 illustrates this approach. Restricting the potential reference points to nodes of a certain colour, we can enforce that the reference point and the

centre of an aggregate are the same. This certainly is a reasonable simplification: it is preferred that nodes 'gather around' a centre of unchanged colour.

It is important to note that just as $c_{\text{perimeter}}$ this measure is not size-invariant: as distances to centres are shorter for small areas, setting $c := c_{\text{distance}}$ will tend to result in solutions with many small aggregates. This bias, however, is limited since the thresholds θ guarantee that the aggregates do not become too small.

As shapes are approximated by centroids, the measure c_{distance} is a rather coarse indicator for the geometrical characteristics of an aggregate. We therefore suggest to combine c_{distance} with the measure $c_{\text{perimeter}}$ by defining an additional weight factor $s' \in [0, 1]$, that is,

$$c := s' \cdot c_{\text{distance}} + (1 - s') \cdot c_{\text{perimeter}}. \tag{4}$$

Finally, we introduce the measure $c_{\text{shortest-path}}$ that is slightly different from c_{distance}. Let $\delta'_{V'}(u, v)$ be the length of the shortest path π from $u \in V'$ to $v \in V'$ on G using edge length δ, such that π does not leave $V' \subseteq V$. We define $c_{\text{shortest-path}}$ by equation (3), but replace δ by $\delta'_{V'}$. This measure is illustrated in Figure 5. For certain problems $c_{\text{shortest-path}}$ is certainly more adequate than c_{distance}, for example, if a traveller is not allowed to cross territory boundaries. However, for map generalisation we consider c_{distance} and $c_{\text{shortest-path}}$ similarly relevant: both can be seen as rather abstract driving forces toward more compact shapes.

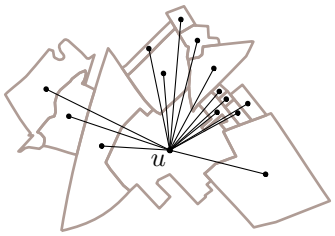


Figure 4.: The compactness measures c_{distance} uses direct distances to a reference point.

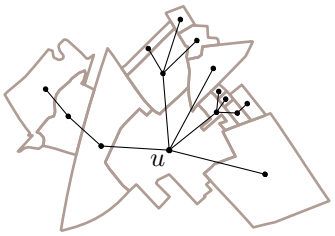


Figure 5.: The compactness measures $c_{\text{shortest-path}}$ uses shortest path lengths to a reference point.

4. NP-hardness

We now investigate the computational complexity of the aggregation problem. We show that the problem is NP-hard, that is, one cannot expect to find an exact polynomial-time solution.

Theorem 4.1 : *Given an instance of* AreaAggregation *defined in Section 3.1 and an integer $C > 0$, it is NP-hard to decide whether a solution with cost less than C exists, even if the number of colours is restricted to two, all nodes have unit weight, the distance d between each two colours is the same, the threshold θ is the same for all classes, and the compactness of shapes is neglected, that is, $s = 1$.*

Note that in this decision version of the aggregation problem, we only look for an answer like 'yes' (there exists a solution better than C) or 'no'. Of course we could easily answer this question by optimally solving the aggregation problem. So, the decision version is not harder than the optimisation version.

Our proof relies on the following concepts.

- Given a graph $G(V, E)$, a *vertex cover* of G is a subset of V that contains at least one of the two endpoints of each edge in E.

- Given a graph G and an integer $C > 0$, the problem VertexCover is to decide whether G has a vertex cover of cardinality at most C.
- The problem PlanarVertexCover is the special version of VertexCover where the given graph G is planar. Even this restriction is known to be NP-hard (Garey *et al.* 1974).

As usual we prove NP-hardness by reduction from a problem that is known to be NP-hard. In our proof we reduce from PlanarVertexCover, that is, we show that, for each instance of PlanarVertexCover, we can construct an instance of the decision version of AreaAggregation such that the second instance is a 'yes'-instance if and only if the first instance is a 'yes'-instance. This means that, if there was an efficient algorithm for the area aggregation problem, we could efficiently solve the problem PlanarVertexCover as well. In other words, solving the area aggregation problem is at least as hard as solving PlanarVertexCover. As PlanarVertexCover is NP-hard, AreaAggregation is NP-hard, too.

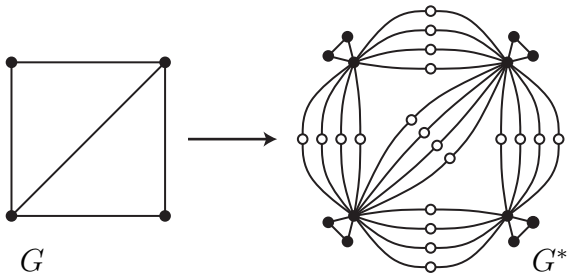


Figure 6.: Reduction from PlanarVertexCover.

Proof: Given an instance of PlanarVertexCover, that is, a planar graph $G(V, E)$ and an integer $C > 0$, we construct the input graph $G^*(V^*, E^*)$ for the aggregation problem as in Figure 6. We define the set of colours $\Gamma = \{\text{black}, \text{white}\}$. For each node $v \in V$, we add three black nodes to V^*. We refer to an arbitrary one of them as the node corresponding to v. The three nodes are connected by edges forming a triangle. For each edge $\{u, v\} \in E$, we add $|V|$ white nodes to V^*. We connect each of these nodes by edges with the nodes corresponding to u and v. We define a unit weight $w = 1$ for each node and the threshold $\theta = 2$ for each colour. For changing the colour of a node, we charge one unit of cost, that is, we define $d(\text{white}, \text{black}) = d(\text{black}, \text{white}) = 1$ and $d(\text{white}, \text{white}) = d(\text{black}, \text{black}) = 0$. The maximally allowed cost is defined by $C^* = C$.

We now prove that there is a solution of the instance of AreaAggregation with cost C^* if and only if there is a vertex cover of G with cardinality C.

For each vertex cover of G with cardinality $\bar{C}$, there is a corresponding solution of the aggregation problem with cost $\bar{C}$: in G^*, we simply need to change the black nodes that correspond to the nodes in the vertex cover of G into white nodes. As the vertex cover includes an endpoint for each edge in E, all white nodes will get a white neighbour. For each triangle of black nodes, at least two adjacent nodes will remain black. Thus the subgraphs induced by the white and by the black nodes are all weight-feasible.

It remains to show that, for any solution of the aggregation problem with cost $\bar{C}$, there is a vertex cover of G with cardinality $\bar{C}$. Let's assume that no such vertex cover exists, that is, the cardinality of a minimum vertex cover of G is larger than $\bar{C}$. This implies that there is at least one edge $\{u, v\}$ in E such that the black nodes in V^* corresponding to u and v keep their colours. In this case, we can only satisfy the weight thresholds by changing the colour of all white nodes in V^* that were added for the edge $\{u, v\}$. This, however, is very expensive, that is, we need to

spend a cost of $|V|$. As $|V|$ is the cardinality of a trivial vertex cover of G, that is, the one including all nodes in V, we have found a contradiction to our assumption.

Note that our reduction can be done in polynomial time. Therefore, we can say that PlanarVertexCover *polynomially reduces to* AreaAggregation. □

Since there is no hope of finding an exact polynomial time algorithm for AreaAggregation, we focus on heuristics and mixed-integer programming.

5. MIP formulation

According to the problem definition from Section 3.1, only contiguous aggregates are allowed. To express this requirement by means of linear constraints is a non-trivial task. Williams (2002) and Shirabe (2005) have found different solutions for this in the context of spatial allocation problems. Their models allow constraints to be defined that forbid non-contiguous aggregates but do not exclude any contiguous aggregate. Both authors discuss problems where the aim is to compute a single contiguous aggregate. In this case, their formulations require a linear number of variables and constraints. The MIPs of both authors can be adopted in order to model the aggregation problem. However, as this problem asks for several contiguous aggregates, the number of variables and constraints becomes quadratic.

We tested both methods for the area aggregation problem; it turned out that only very small problem instances can be solved optimally. The largest instance that we could solve optimally contained 15 nodes with the approach of Williams (2002) and 30 nodes with the approach of Shirabe (2005). However, for the special cases that compactness is either neglected or expressed by $c_{\text{shortest-path}}$, we found a MIP of linear size. With this model we solved instances of up to 50 nodes. For the MIP formulation we refer to Appendix B. We refer to this MIP as *flow MIP*.

In order to obtain an appropriate performance we introduce heuristics. In particular, we introduce a stricter requirement for the contiguity of aggregates. This approach has been used before for spatial allocation problems (Zoltners and Sinha 1983) and districting problems (Caro *et al.* 2004). Both works deal with problems where a set of region centres is given in advance. In contrast, we need to solve the aggregation problem for an unknown set of centres. We first specify the approach and introduce a basic MIP that allows for the compactness measure c_{distance} (Section 5.1). We then extend this MIP to also consider $c_{\text{perimeter}}$ (Section 5.2).

5.1. *A basic MIP based on a precedence relationship*

In order to present our MIP formulation we first define the set of variables:

$$x_{uv} \in \{0,1\}, \quad \text{with } x_{uv} = 1 \text{ if and only if node } v \in V \text{ belongs to centre } u \in V.$$

These binary variables define the solution of the problem: all nodes u with $x_{uu} = 1$ constitute the set of centres that define the colours of aggregates and the reference points for the compactness measure c_{distance}. To assign a node v to a centre u, x_{uv} needs to be set to 1. We minimise the cost f_{recolour} by the objective function

$$\text{minimise} \sum_{u \in V} \sum_{v \in V} w(v) \cdot d(\gamma(v), \gamma(u)) \cdot x_{uv}. \tag{5}$$

Similarly, we minimise the cost for the non-compactness according to c_{distance}:

$$\text{minimise} \sum_{u \in V} \sum_{v \in V} w(v) \cdot \delta(u, v) \cdot x_{uv} \tag{6}$$

We now explain our set of constraints. The first constraint expresses that each node must be assigned to exactly one centre:

$$\sum_{u \in V} x_{uv} = 1 \quad \text{for all } v \in V \tag{7}$$

The second constraint ensures that the aggregate with centre u is weight-feasible, that is, the threshold for the target scale is satisfied. It is only effective if $x_{uu} = 1$.

$$\sum_{v \in V} w(v) \cdot x_{uv} \geq \theta\big(\gamma(u)\big) \cdot x_{uu} \quad \text{for all } u \in V \tag{8}$$

It remains to ensure the contiguity of aggregates. For this we first define the set of predecessors of a node v with respect to the centre u by

$$\text{Pred}_u(v) := \{w \in V \mid D(u, w) < D(u, v) \wedge \{v, w\} \in E\}, \tag{9}$$

where $D : V \times V \to \overset{+}{0}$ denotes a distance between nodes. According to this definition, the set $\text{Pred}_u(v)$ contains the neighbours of v that are closer to u.

We define the distance D to be the shortest-path length in the directed graph $G'(A, V)$ with $A = \{uv \mid \{u, v\} \in E\}$ and arc lengths $\alpha : A \to \overset{+}{0}$. Furthermore, we define $\alpha(uv) = w(v)$, which implies that we consider two nodes to be far from each other if the smallest aggregate that contains both is large.

Figure 7(a) illustrates the defined precedence relationship assuming unit node weights. Arcs are drawn from each node $v \in V$ to its predecessors $\text{Pred}_u(v)$. The resulting directed graph is acyclic and the centre u is the only sink. For some edges (dashed lines), both incident nodes have the same distance to the centre. With non-uniform node weights, however, these cases are very rare.

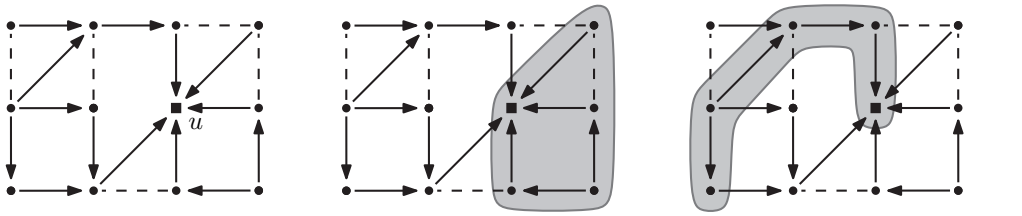
We now ensure contiguity by claiming that node v can only be assigned to centre u if there is also a predecessor w that is assigned to u:

$$\sum_{w \in \text{Pred}_u(v)} x_{uw} \geq x_{uv} \quad \text{for all } u, v \in V \text{ with } u \neq v. \tag{10}$$

This constraint clearly forbids non-contiguous aggregates since the centre can always be reached from an assigned node via predecessors, that is, without leaving the aggregate. Figure 7(b) shows an example that satisfies the constraint. Clearly, when applying this constraint, some contiguous aggregates are excluded, for example, the aggregate in Figure 7(c). The aggregate does not contain any predecessor of the node located in the bottom left corner. This strict definition of contiguity is justifiable, since the non-feasible contiguous aggregates are likely to be non-compact. Therefore, we can probably exclude them without loosing good solutions. Since the optimal solution might be missed, we refer to this approach as a heuristic.

5.2. *Expressing the cost for perimeters*

To express the cost for perimeters of shapes, we need additional auxiliary variables:



(a) the precedence relationship Pred_u (b) aggregate feasible according to Pred_u (c) aggregate non-feasible according to Pred_u

Figure 7.: Precedence relationship with respect to centre u (displayed as square) and feasibility of aggregates in the presented MIP.

$$y_{ue} \in [0,1], \quad \begin{array}{l}\text{with } y_{ue} = 0 \text{ if at least one endpoint of } e \in E \\ \text{does not belong to centre } u \in V.\end{array}$$

We add the following constraint to ensure the above semantic of the variables y_{ue}, that is, y_{ue} with $e = \{v, w\}$ is set to 0 if $x_{uv} = 0$ or $x_{uw} = 0$:

$$\left.\begin{array}{rl} y_{ue} \leq & x_{uw} \\ y_{ue} \leq & x_{uv} \end{array}\right\} \quad \text{for all } u \in V, e = \{v, w\} \in E \tag{11}$$

Charging a cost for the perimeters of aggregates can also be regarded as giving a benefit for edges contained in aggregates. With the variables y_{ue} we can choose this approach. Let's first express this alternative view for the total cost for perimeters. The sum of perimeters in the result is equal to the sum of perimeters in the input minus twice the length of boundaries that are eliminated by the aggregation:

$$\sum_{V_i \in P} c_{\text{perimeter}}(V_i) = \sum_{v \in V} c_{\text{perimeter}}(\{v\}) - 2 \cdot \sum_{e \in E'} \lambda(e), \tag{12}$$

with $E' \subseteq E$ being the set of edges that are contained in an aggregate and $\lambda : E \to \overset{+}{0}$ being the length of the common boundary between two areas. As the first term on the right-hand side of equation (12) is constant, we only need to minimise the second term. This can be expressed by

$$\text{minimise } -2 \cdot \sum_{u \in V} \sum_{e \in E} \lambda(e) \cdot y_{ue}. \tag{13}$$

As a negative cost, that is, a benefit is given proportional to y_{ue}, the variable y_{ue} always attains the value of its upper bound. Hence $y_{ue} = 1$ for each edge $e \in E$ included in the aggregate with centre u. We keep the factor 2 in objective (13). Thus, we can express the cost $f = s \cdot f_{\text{recolour}} + (1 - s) \cdot f_{\text{non-compact}}$ (see (vii)) with the compactness function $c = s' \cdot c_{\text{distance}} + (1 - s') \cdot c_{\text{perimeter}}$ (equation (4)) simply by applying the same factors s and s' to objectives (5), (6), and (13).

6. Heuristic approaches

In Section 5.1 we already introduced a heuristic by excluding certain contiguous aggregates in advance. However, this does not suffice to obtain an adequate performance. In this section we suggest additional specialised heuristics (Sections 6.1 and 6.2) and an alternative approach by simulated annealing (Section 6.3).

In our earlier publication (Haunert and Wolff 2006) we defined a heuristic, which we termed *distance heuristic*. This heuristic is based on the fact that it is very unlikely that two nodes u and v are merged in the same aggregate if their distance is large. More precisely, we set $x_{uv} = x_{vu} = 0$ if a certain, easily accessible predicate allows us to conclude that each contiguous aggregate containing u and v can be split into two feasible aggregates. In this article, we skip the formal definition of this predicate since the distance heuristic only led to a small reduction of the processing time. Nevertheless, we present experimental results with this heuristic in Section 7.

6.1. *Centre heuristic*

A very common approach to speed up the solution of districting problems is to predefine the set of centres (Hess and Samuels (1971), Hojati (1999)). Shirabe (2005) shows that his MIP for selecting an optimal contiguous set of areas can be simplified if a centre is known. We fix the set of centres to reduce the number of variables in our problem instances. Our idea is to define heavy nodes as centres, as the weight of a node is a multiplier for costs that are charged for its colour change and its distance to the centre. So, it is likely that solutions with large centres are good according to the objective function. Consequently, we exclude nodes with low weights from the set of potential centres.

When defining a set of nodes as centres we need to ensure that the problem does not become over-constrained. We can guarantee the feasibility of the problem if we maximally fix one node with unchanged colour for each aggregate in a feasible start solution. To find a start solution we apply Algorithm 1. As the quality of this solution is relatively low, we choose a rather conservative approach, that is, the majority of nodes will not be constrained. We define the centre heuristic as follows.

Centre Heuristic: *(a) For each aggregate obtained with Algorithm 1, the largest area with unchanged colour is a centre and (b) each other area of size less than 10% of the threshold is excluded from the set of potential centres.*

With this definition, all areas in the original scale that are sufficiently large for the target scale will be included in the set of predefined centres. About 7% of all areas in our dataset at scale 1:50,000 fall into this category when applying the specifications for the target scale 1:250,000. Predefining these nodes as centres does not necessarily need to be regarded as deficit. Recall our notion of completeness from Section 1.2: an area that is large enough for the target scale must not be lost, that is, its class must not change. This is ensured by fixing such nodes as centres. However, also other nodes will be constrained with the defined heuristic, which is necessary to obtain an acceptable performance.

It is clear that certain variables can immediately be eliminated with this heuristic. Additionally, the heuristic allows a problem instance to be decomposed into smaller instances as follows.

Proposition 6.1: *Let $V_\theta = \{v \in V \mid w(v) \geq \theta(\gamma(v))\}$. Under the condition that each node in V_θ is fixed as a centre and the subgraph of G induced by $V \setminus V_\theta$ is not connected, the* AreaAggregation *instance with any of the compactness measures c_{distance}, $c_{\text{shortest-path}}$, and $c_{\text{perimeter}}$ decomposes into several (that is, two or more) independent problem instances. Each such problem instance comprises a connected component of the subgraph of G that is induced by nodes $V \setminus V_\theta$ and the centres surrounding this component.*

In the example in Figure 8, the problem can be decomposed into two instances. The proposition directly follows from the definition of the problem: suppose that

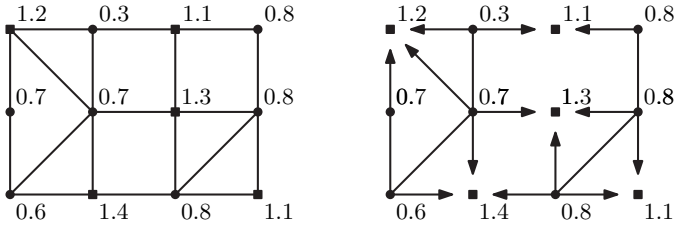
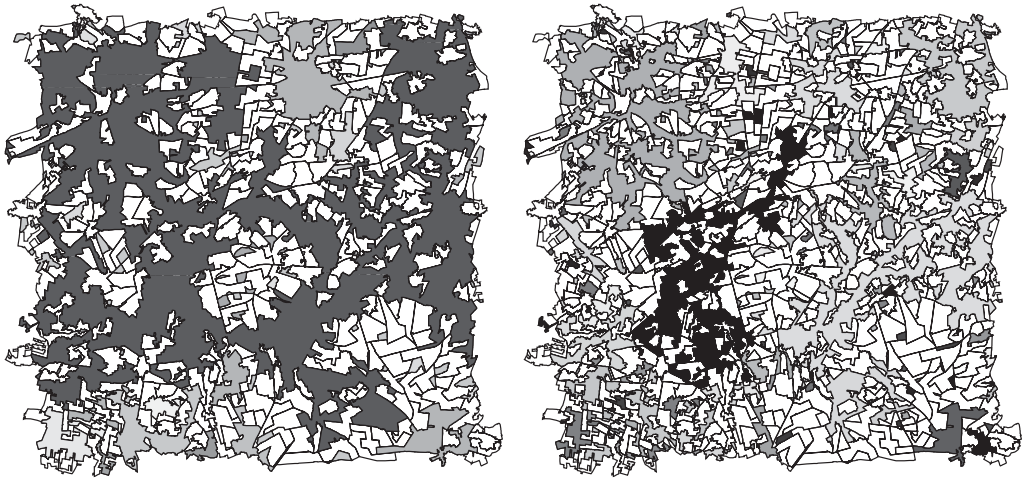


Figure 8.: An instance of the aggregation problem (left), which can be decomposed into two independent problem instances (right). Node weights are displayed as numbers, centres with $w(v) = \theta(\gamma(v))$ as squares. The threshold is defined by $\theta(\gamma) = 1$ for all $\gamma \in \Gamma$.



(a) applying thresholds for target scale 1:250,000 (b) applying half thresholds

Figure 9.: Independent problem instances resulting from the centre heuristic (different grey shades). The largest problem instance is displayed with the darkest shade. White areas have sufficient size.

two nodes i and j belong to different connected components. Then each potential aggregate containing both also contains a fixed centre v with $w(v) \geq \theta(\gamma(v))$. The assignment of i to v does not have any influence on the cost of the assignment of j to v. Further, as v has already sufficient weight, these merges do not have any influence on the feasibility of the aggregate containing v.

6.2. *Introducing intermediate scales*

Figure 9 shows the independent problem instances resulting from the centre heuristic for our dataset at scale 1:50,000, which contains 5537 areas. In Figure 9(a) we applied the thresholds from the existing specifications for the target scale 1:250,000. Though the problem decomposes into 145 instances, one huge connected component remains that contains 4226 areas; this is far too much to be processed at once. We therefore suggest to define intermediate scales, that is, to satisfy smaller thresholds first. In Figure 9(b) we applied thresholds of half value, yielding much smaller independent problem instances. If we know that we can solve instances with at most K nodes, we can further decrease the thresholds until no instance is larger than this. Hence, the capability to solve problem instances of limited size can be translated into the capability to handle limited differences in scale.

Certainly, the number of scale steps that are needed to reach the target scale is different for different parts of the dataset: our dataset contains large forest areas that have a separating effect. In the surroundings of towns, however, we find many small areas. We therefore suggest Algorithm 2 that locally introduces intermediate thresholds whenever they are needed to restrict the size of the resulting instances.

Figure 10 illustrates the algorithm using a one-dimensional example.

The algorithm requires the parameter K as input, that is, the maximal number of areas that are to be processed in one iteration. Setting $K := 1$ the algorithm iteratively selects the smallest area and assigns it to one of its neighbours, which is just the same as what is done by the original iterative algorithm in Section 1.1. Our assumption is that we obtain better results if we choose K as high as possible, in a way allowing the optimisation process to think more merges ahead. If we set $K := |V|$, we solve the problem in a single step without intermediate thresholds. Thus, the algorithm generalises both: the original iterative approach as well as the optimisation approach with the centre heuristic.

Throughout the algorithm, we maintain a set of connected components (defined in line 2), which we let grow by iteratively adding the smallest area that falls below the required threshold (lines 4–21). If we obtain a connected component containing K areas, we define and solve a problem instance as shown in Figure 8. This means that we allow the algorithm to aggregate the contained areas with each other or to assign them to one of the adjacent areas (line 12). Then we update the map by replacing the areas with the aggregates we found. We need to define the intermediate thresholds θ' for this step such that all areas in the neighbourhood of the component have sufficient weight; thus we define $\theta'(\gamma) = \min\{\theta(\gamma), w_{\min}\}$, with $w_{\min}$ being the smallest area in the neighbourhood.

If different components merge when adding a single area (lines 17–18), instances of more than K areas may come into existence. To avoid this, we solve the largest involved component. Finally, we solve the remaining instances applying the thresholds for the target scale (line 22).

6.3. *Area aggregation by simulated annealing*

In order to specify our alternative heuristic optimisation approach by simulated annealing, we adhere to the common definition of the algorithm. For pseudo code and a list of required design decisions we refer to Johnson *et al.* (1989) who applied simulated annealing to graph partitioning. We choose the following setting:

- We apply Algorithm 1, that is, region growing, to find an initial solution.
- We define the neighbourhood of a solution as the set of solutions that can be obtained by applying a single *node swap*.

A node swap comprises the removal of a node from its aggregate and the assignment of this node to another aggregate. Figure 11 shows different cases. In any case, we recolour the involved aggregates such that the resulting cost is minimal.

A node swap can result in an aggregate whose size falls below the required threshold θ. We could reject such solutions, but as a consequence we would lose the possibility of reaching any solution from a given start, for example, a solution that only contains aggregates of size θ would be an isolated point in the solution space. Hence, we relax the hard threshold constraint and charge an additional cost for an aggregate $V' \in P$ with colour γ', if its size is smaller than $\theta(\gamma')$. We define this cost by $s'' \cdot \left(\theta(\gamma') - \sum_{v \in V'} w(v)\right)$ with a new weight factor $s'' \in \overset{+}{0}$. If the final result of the algorithm does not satisfy the size constraints, we can apply Algorithm 1 to repair the solution.

We define the annealing schedule by the initial temperature T_0, the final temperature T_E, and the number of iterations I. The cooling ratio that defines the temperature decrease in one iteration is $r = (T_E/T_0)^{1/I}$. For our experiments we set $T_0 = 10^9$, $T_E = 10^4$, and $s'' = 10^4$ (areas were measured in m^2 and distances in m). We set $I = 1000 \cdot |V|$ if we do not explicitly define another setting.

Algorithm 2 Iterative aggregation of areas in big scale steps

1: Input: an instance of AreaAggregation, an integer $K > 0$
2: $\Pi \leftarrow$ a set of connected components, initially empty
3: $S \leftarrow$ set of areas below threshold for target scale
4: **while** $S \neq \emptyset$ **do**
5: $a \leftarrow$ smallest area in S
6: $\Pi' \leftarrow$ the set of components in Π containing a neighbour of a
7: $S' \leftarrow$ the set of areas that lie in one of the components in Π'
8: **if** $|S'| < K$ **then**
9: Remove all components in Π' from Π.
10: Create a new component p comprising a and all areas in S'.
11: **if** p contains K areas **then**
12: Solve p: merge areas in p with each other or with surrounding centres.
13: **else**
14: Insert p into Π.
15: **end if**
16: **else**
17: Solve the component in Π' having most areas as in line 12.
18: Remove this component from Π.
19: **end if**
20: $S \leftarrow$ set of areas below threshold for target scale, not contained in any component in Π
21: **end while**
22: Solve the remaining components in Π as in line 12.

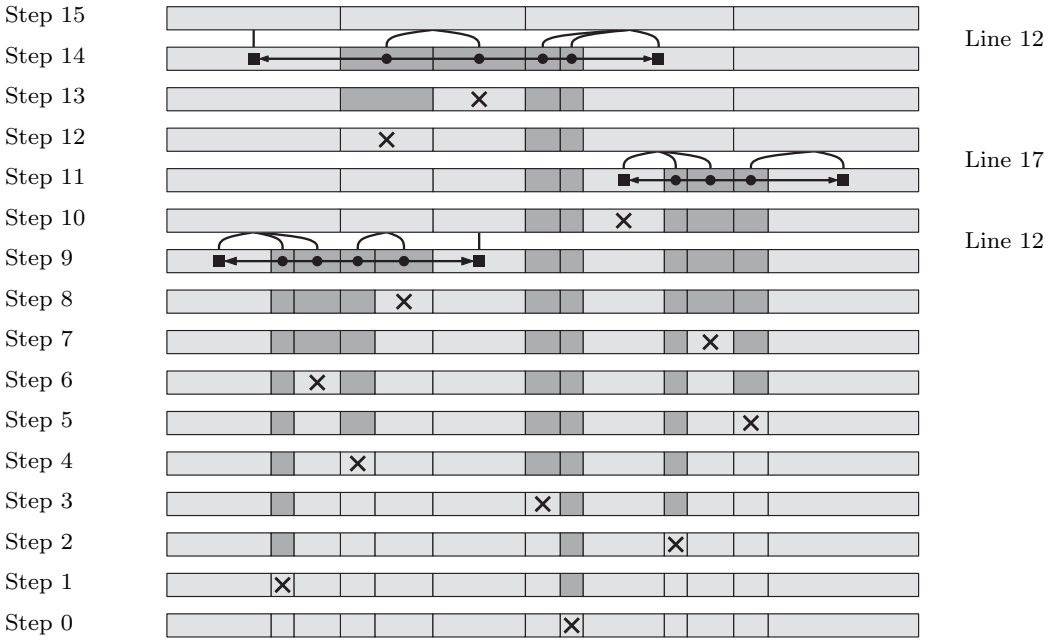
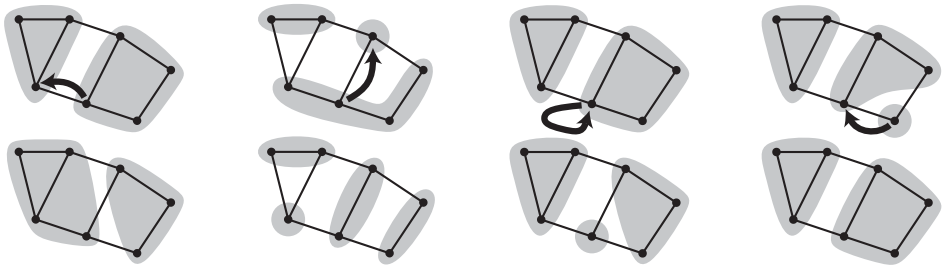


Figure 10.: Several steps of Algorithm 2 with $K = 4$ (from bottom to top). The optimisation method was applied three times; both cases that are defined in lines 12 and 17 of the algorithm occurred. The meaning of the small disks and squares is as in Fig. 8. Areas that were added to components in Π are displayed as dark grey rectangles. Among the other areas (light grey rectangles), the smallest area is selected in each iteration (marked with $\times$). The connected components in Π are updated by adding this area, if this does not imply a connected component of more than four areas. If a connected component of exactly four areas is created, the corresponding problem instance is solved. In the example, this happens in steps 9 and 14. If adding an area would imply a connected component with more than four areas, the problem instance corresponding to the largest adjacent component is solved. In the example, this happens in step 11: adding the smallest area in step 10 to Π would imply a connected component of six areas.



(a) the normal case, before (top) and after application of the node swap

(b) removing a node, an aggregate resolves into multiple contiguous aggregates

(c) we also allow a new aggregate to be created with a single node

(d) an aggregate with a single node completely merges into another one

Figure 11.: Different types of node swap that define a neighbourhood for simulated annealing.

7. Experimental tests

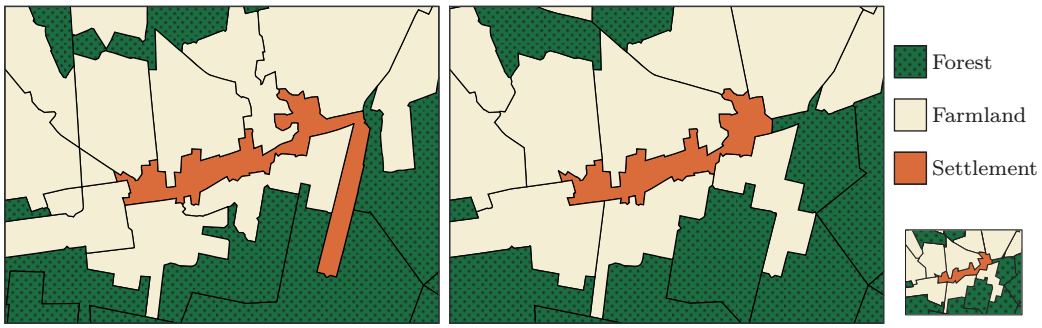
We tested our methods for a dataset from the ATKIS database at scale 1:50,000, applying size thresholds from existing specifications for the scale 1:250,000. The class distances d were defined according to Table 1. All remaining parameters of our model were found by experiments. We first present some samples and then discuss the observed performance. We present a large sample in Appendix C.

7.1. *Obtained results*

Applying the method from Section 6.2 with intermediate scales, we are able to process large datasets. We used this approach to produce the presented results, that is, we applied Algorithm 2 with $K = 200$ and solved the occurring problem instances with the MIP based on the precedence relationship. Both the centre and the distance heuristic were applied. For three sample of our dataset we show:

- the input dataset after preprocessing with the collapse procedure (Figures 1(a), 13(a), and 14(a)),
- the result of Algorithm 1, i.e., region-growing (Figures 1(b), 13(b), and 14(b)),
- the result of the optimisation method when minimising class change (Figures 12(a), 13(c), and 14(c)),
- the result of the optimisation method when minimising the combined cost with $s = 0.000715$ and $s' = 0.000015$ (Figures 12(b), 13(d), and 14(d)), and
- the latter aggregation result after line simplification, shown at the target scale 1:250,000 (Figures 12(c), 13(e), and 14(e)).

Figure 12 shows the results we obtained for the input in Figure 1(a). In Figure 12(a) we minimised the cost for class change. In contrast to Algorithm 1, this optimisation approach allows small settlements to be saved by changing the classes of some connecting areas. Though this leads to an aggregate of sufficient size for the target scale, the example clearly shows that minimising changes of classes does not suffice to produce good generalisation results: the small rectangular settlement in the right part of Figure 1(a) was included in the aggregate by creating a long, narrow corridor that is needed to satisfy the constraint for connectivity; obviously such complex shapes are not intended. Minimising a combined cost for class change and compactness, we obtained the result in Figure 12(b). Including the rectangular settlement in the aggregate would be too expensive, as the resulting boundary would be relatively long. As a consequence, we obtain a more compact aggregate that better reflects the aim of a cartographer.



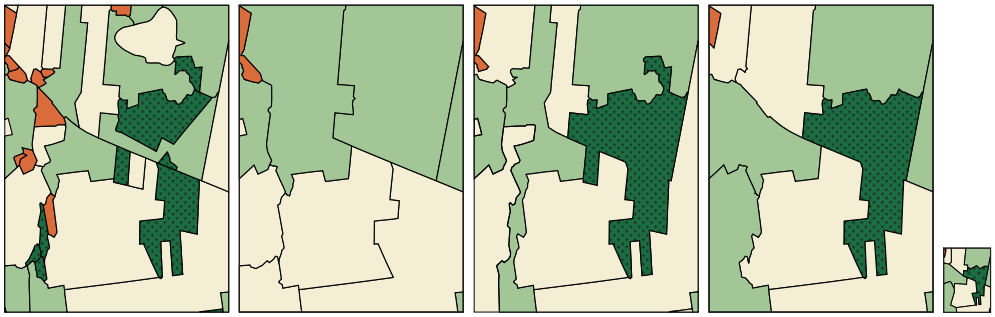
(a) result with minimal class change

(b) considering class change and compactness

(c) at scale 1:250000 (lines simplified)

Figure 12.: Two results for the sample in Figure 1(a) obtained with our optimisation approach.

A similar example is presented in Figure 13. The forest areas in the right part of Figure 13(a) are lost when applying the simple iterative method (Figure 13(b)), but can be saved using our optimisation approach. The results without and with consideration of compactness (Figure 13(c) and 13(d), respectively) mainly differ in the left part of the sample. However, also the forest in the right part appears slightly different in both results. In Figure 13(d) two small forest areas were 'sacrificed', that is, changed to other classes, in order to create a short and simple outline.



(a) input

(b) output of Alg. 1

(c) as Fig. 12(a)

(d) as Fig. 12(b)

(e) as Fig. 12(c)

Figure 13.: A second sample from the ATKIS DLM 50.

(a) input

(b) output of Alg. 1

(c) as Fig. 12(a)

(d) as Fig. 12(b)

(e) as Fig. 12(c)

Figure 14.: A settlement that can be saved by 'stealing' smaller forest areas.

The sample in Figure 14 shows another interesting case. The settlement in the centre is too small for the target scale, but can be saved either by creating a con-

Table 2.: Experimental results with our MIPs, neglecting compactness. Computation times are in seconds CPU time unless marked with **h**, which stands for hours. All instances were solved to optimality except *. The average class distance $\bar{d}$ is equal to the cost for class change per area as defined in equation (1). Class distances are defined according to Table 1.

	Alg. 1	flow MIP				P-R MIP							
		pure		centre heuristic		pure		distance heuristic		centre heuristic		centre + dist. heuristic	
nodes	$\bar{d}$	time	$\bar{d}$	time	$\bar{d}$	time	$\bar{d}$	time	$\bar{d}$	time	$\bar{d}$	time	$\bar{d}$
30	29.30	223.2	9.20	8.8	9.20	2.8	9.20	3.0	9.20	0.04	12.81	0.01	12.81
40	22.31	3.9 **h**	6.97	41.6	6.97	31.1	7.34	16.8	7.34	0.04	7.59	0.03	7.59
50	15.30	*20.0 **h**	5.18	432.1	5.18	60.9	5.18	62.5	5.18	0.34	5.64	0.15	5.64
60	15.49					253.0	5.85	236.0	5.85	0.45	6.66	0.72	6.66
100	12.48									33.1	5.93	12.5	5.93
200	9.85									84.3	4.68	87.0	4.68
300	6.92									267.2	4.50	363.3	4.50
400	7.04									397.7	4.64	346.1	4.64

nection to another settlement (Figure 14(c)) or by including small adjacent forest areas (Figure 14(d)). This latter result reflects the idea of exaggerating important map features, which is a common approach in map generalisation.

7.2. *Performance tests*

In this section we present the numerical results of our tests, that is, the running time of our methods and the quality of the obtained solutions according to the applied cost functions. For all our tests we used a Linux server with 4 GB RAM and a 2.2 GHz AMD-CPU. For the solution of our MIPs we used the Interactive Optimizer and the Java interfaces of the software ILOG CPLEX™ 11.2.

We tested both the flow MIP and the MIP from Sections 5.1 and 5.2 based on the precedence relationship (referred to as *P-R MIP*) on instances of different sizes and with different combinations of the presented heuristics. The flow MIP offers optimal solutions for small instances, but does not allow us to apply the same measures of compactness as the P-R MIP (in the flow MIP, the assignment of an area to its centre is encoded only implicitly via the flow). Hence, to compare the performance, we only consider the objective for class change in this test.

Table 2 summarises our results. All instances were solved with proof of optimality, except the one marked with a star. The first column of Table 2 shows the results that were obtained with the iterative Algorithm 1. With this method, all computations took less than a second. The time needed for the solution of our flow MIP is very high, but the results allow us to assess the results of the other methods. The computation time can be reduced with the centre heuristic from Section 6.1. On our examples the heuristic yielded the optimum. By definition of the precedence relationship the computation time is reduced while the solution is only marginally affected. When we applied the centre heuristic to the P-R MIP, the running time was decreased drastically, whereas the quality of the results was not affected much. We also processed the same instances with an older version of the optimisation software, that is, ILOG CPLEX™ 9.1. We listed the results in our earlier publication (Haunert and Wolff 2006). Applying the more recent version to our largest problem instances reduced the processing time by more than 60%.

With the same setting for the objective function we tested the performance of simulated annealing (Table 3). Our tests show that the quality of the results is similar to those obtained with the P-R MIP in combination with the centre and distance heuristic. For small instances, we observed a clearly better performance with our approach by mixed-integer programming. For the instance with 400 nodes, our simulated annealing method resulted in a solution of 15.3% more costs, but required much less time than needed to solve the MIP.

Table 3.: Experimental results with our approach by simulated annealing. Computation times are in seconds CPU time unless marked with **min**, which stands for minutes. Except for the instance of 5537 nodes, the problem instances are the same as in Table 2. The instance of 5537 nodes corresponds to our whole dataset. For this instance we defined the number of iterations such that we expected a running time similar to that of Algorithm 2 with $K = 200$, see column 3 of Table 4.

nodes	iterations	time	$\bar{d}$
30	30000	3.6	15.27
40	40000	3.8	7.08
50	50000	4.1	6.63
60	60000	4.7	6.39
100	100000	9.1	9.05
200	200000	18.9	6.42
300	300000	25.2	4.85
400	400000	32.9	5.35
5537	12500000	23.35 **min**	4.33

Table 4.: Experimental results with Algorithm 2. The processed dataset covers an area of 22 km × 22 km and contains 5537 polygons (nodes). The parameter K defines the maximal number of nodes v with $w(v) < \theta(\gamma(v))$ that are processed in one iteration of Algorithm 2 by applying the P-R MIP with heuristics. Objective values (costs) are normalised with respect to the results for $K = 1$. Results are shown for tests without consideration of compactness (columns 2–3) and with it (columns 4–8).

optimisation criteria	class change		class change and compactness				
K	1	200	1	50	100	150	200
time (minutes CPU time)	1.12	23.47	1.21	1.99	9.44	24.43	58.57
$\bar{d}$	6.09	4.54	6.54	5.90	5.58	5.58	5.24
cost for class change	100	74.5	100	90.2	85.3	85.3	80.1
cost for non-compactness	–	–	100	99.3	98.1	97.8	98.8
total cost	100	74.5	100	96.3	93.9	93.7	92.6

As the method by mixed-integer programming with the distance and the centre heuristic is too slow to process large datasets, that is, thousands of polygons, we also tested our method with intermediate scale steps. Our basic assumption was that we obtain results of higher quality if we apply the iterative method, considering more than one area with its direct neighbours in each step. To verify this assumption, we applied Algorithm 2 with different settings of the parameter K, see Table 4.

In a first test we solved our instance of 5537 nodes with the objective of minimal class change. Setting $K = 1$ we obtained a solution with $\bar{d} = 6.09$. With this setting our method produces the same result as the iterative algorithm that selects the most compatible neighbour according to the defined cost function. The average class distance was reduced to $\bar{d} = 4.54$ when setting $K = 200$, which is a reduction by 25.5%. In this case the processing took 23.47 minutes. We processed the same instance with our simulated annealing approach, defining to terminate after 12,500,000 iterations, since we expected a similar running time with this setting (last row of Table 3). The actual running time was 23.35 minutes; the average class distance of the solution was $\bar{d} = 4.33$. This test shows that both methods perform similarly for large datasets, however, simulated annealing reduces the costs by 4.6% compared to the deterministic method.

In a second test we considered both objectives: class change and compactness. The results in Figures 12(b), 13(d), and 14(d) were created with this setting. We list the average class distance $\bar{d}$ as well as the costs for class change, non-compact shapes, and their sum (normalised to 100). The resulting costs for non-compactness are very similar for different values of K. It seems that, concerning this objective, the iterative algorithm does quite well by greedily choosing a neighbour. However, the decrease of costs for class change is still considerable (19.9% for $K = 200$). Combining both objectives, we obtain an improvement by 7.4%. These tests clearly confirm the assumption that we obtain results of higher quality if we consider more areas in one step. Consequently, we suggest to set the parameter K as high as possible under the given time constraints.

Comparing the average class distance for both experiments with $K = 200$ we see that, in order to produce reasonably compact shapes, we need to accept an increase of $\bar{d}$ by 15.4%.

8. Conclusion

We have proposed new exact and heuristic methods for automated area aggregation in planar partitions. Our methods produce logically consistent results with respect to given database specifications. Subject to these constraints our methods optimise semantic accuracy and compactness of shapes.

Our tests have revealed that our exact method, the flow MIP, can be used to process small datasets only, that is, instances of up to 50 areas. Still, our MIP formulations, the flow MIP and the P-R MIP, yield the following. First, they can be used to measure the quality of heuristics. Second, they can be combined with heuristics. For example, combining the P-R MIP with other heuristic approaches (such as the very effective centre heuristic) we can solve instances of 400 areas in reasonable time. At least on small instances, costs remained close to optimal. Third, using our method of intermediate scale steps, we could solve large instances of more than 5000 areas. With respect to region growing, our method decreased costs by 25.5% when minimising class change, and by 7.4% when taking compactness into account.

We have shown that, for small instances, mixed-integer programming outperforms simulated annealing both in terms of costs and computation time. On the other hand, when generalising a large dataset from scale 1:50,000 to 1:250,000 with the aim of minimising class change, simulated annealing yielded, within roughly the same time, a solution that was 4.5% cheaper than that of the intermediate-scale method. Still, simulated annealing has a number of disadvantages, which we have discussed. For example, it is unclear whether the annealing parameters we have specified are of any use for generalising maps between two other scales.

It would be very interesting to find ways to integrate our collapse, aggregation and line simplification methods into an overall strategy for the map generalisation problem, also comprising other operators like typification and enlargement.

References

AdV, 2003. *ATKIS-Objektartenkatalog* [online]. Available from: http://www.atkis.de [Accessed 26 November 2008].

Afflerbach, S., Illert, A. and Sarjakoski, T., 2004. The harmonisation challenge of core national topographic data bases in the EU-project GiMoDig. *In*: Vol. XXXV, Part B4:IV of *the International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*. ISPRS, 129–134.

Bard, S., 2004. Quality assessment of cartographic generalization. *Transactions in GIS*, 8 (1), 63–81.

Boffet, A. and Serra, S.R., 2001. Identification of spatial structures within urban blocks for town characterisation. *In*: Vol. 3 of *Proc. 20th International Cartographic Conference (ICC'01)*, Beijing, China, 6–10 August 2001. ICA, 1974–1983.

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.

Caro, F., Shirabe, T., Guignard, M. and Weintraub, A., 2004. School redistricting:

embedding GIS tools with integer programming. *Journal of the Operational Research Society*, 55 (8), 836–849.

Cheng, T. and Li, Z., 2006. Toward quantitative measures for the semantic quality of polygon generalization. *Cartographica*, 41 (2), 487–499.

Cloonan, J.B., 1972. A note on the compactness of sales territories. *Management Science*, 19 (4), 469–470.

Dantzig, G.B., 1963. *Linear Programming and Extensions*. Princeton, NJ: Princeton University Press.

de Berg, M., van Kreveld, M. and Schirra, S., 1998. Topologically correct subdivision simplification using the bandwidth criterion. *Cartography and Geographic Information Systems*, 25 (4), 243–257.

Frank, R. and Ester, M., 2006. A quantitative similarity measure for maps. *In*: A. Riedl, W. Kainz and G. Elmes, eds. *Progress in Spatial Data Handling*. Berlin, Germany: Springer Verlag, 435–450.

Galanda, M., 2003. Automated polygon generalization in a multi agent system. Thesis (PhD). Department of Geography, University of Zurich, Switzerland.

Garey, M.R., Johnson, D.S. and Stockmeyer, L., 1974. Some simplified NP-complete problems. *In: Proc. 6th Annual ACM Symposium on Theory of Computing (STOC'74)*, Seattle, WA, 30 April – 2 May 1974. ACM, 47–63.

Haunert, J.H. and Sester, M., 2008. Area collapse and road centerlines based on straight skeletons. *GeoInformatica*, 12 (2), 169–191.

Haunert, J.H. and Wolff, A., 2006. Generalization of land cover maps by mixed integer programming. *In: Proc. 14th Annual ACM International Symposium on Advances in Geographic Information Systems (GIS'06)*, Arlington, VA, 11–16 November 2006. ACM, 75–82.

Hess, S.W. and Samuels, S.A., 1971. Experiences with a sales districting model: criteria and implementation. *Management Science*, 18 (4, Part II), 41–54.

Hojati, M., 1999. Optimal political districting. *Computers & Operations Research*, 23 (12), 1147–1161.

Jaakkola, O., 1997. Quality and automatic generalization of land cover data. Thesis (PhD). Department of Geography, University of Helsinki, Finland.

Johnson, D.S., Aragon, C.R., McGeoch, L.A. and Schevon, C., 1989. Optimization by simulated annealing: an experimental evaluation; part I, graph partitioning. *Operations Research*, 37 (6), 865–892.

Karmarkar, N., 1984. A new polynomial-time algorithm for linear programming. *Combinatorica*, 4 (4), 373–396.

Kirkpatrick, S., Gelatt, C.D.Jr. and Vecchi, M.P., 1983. Optimization by simulated annealing. *Science*, 220 (4598), 671–680.

MacEachren, A.M., 1985. Compactness of geographic shape: comparison and evaluation of measures. *Geografiska Annaler. Series B, Human Geography*, 67 (1), 53–67.

Michalewicz, Z. and Fogel, D.B., 2004. *How to Solve It, Modern Heuristics*. Berlin, Germany: Springer Verlag.

Mitchell, J.E., 2002. Branch-and-cut algorithms for combinatorial optimization problems. *In*: P.M. Pardalos and M.G.C. Resende, eds. *Handbook of Applied Optimization*. Oxford, UK: Oxford University Press, 65–77.

Morrison, J.L., 1995. Spatial data quality. *In*: S.C. Guptill and J.L. Morrison, eds. *Elements of spatial data quality*. Oxford, UK: Elsevier Science, chap. 1, 1–12.

Podrenek, M., 2002. Aufbau des DLM50 aus dem Basis-DLM und Ableitung der DTK50 – Lösungsansatz in Niedersachsen. *Kartographische Schriften, Band 6, Kartographie als Baustein moderner Kommunikation*. Bonn, Germany: Kirschbaum Verlag, 126–130.

Regnauld, N., 2001. Constraint based mechanism to achieve automatic generalisation using agent modelling. *In: Proc. GIS Research UK 9th Annual Conference (GISRUK'01)*, University of Glamorgan, UK, 18–20 April 2001. 329–332.

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

Sester, M., 2005. Optimization approaches for generalization and data abstraction. *International Journal of Geographical Information Science*, 19 (8–9), 871–897.

Shirabe, T., 2005. A model of contiguity for spatial unit allocation. *Geographical Analysis*, 37, 2–16.

Steiniger, S. and Weibel, R., 2007. Relations among map objects in cartographic generalization. *Cartography and Geographic Information Science*, 34 (3), 175–197.

Timpf, S., 1998. Hierarchical structures in map series. Thesis (PhD). Technical University Vienna, Austria.

van Oosterom, P.J.M., 1995. 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, eds. *GIS and Generalization – Methodology and Practice*. London, UK: Taylor & Francis.

van Smaalen, J.W.N., 2003. Automated aggregation of geographic objects. Thesis (PhD). Wageningen University, The Netherlands.

Ware, J.M. and Jones, C.B., 1998. Conflict reduction in map generalization using iterative improvement. *GeoInformatica*, 2 (4), 383–407.

Ware, J.M., Jones, C.B. and Thomas, N., 2003. Automated map generalization with multiple operators: a simulated annealing approach. *International Journal of Geographical Information Science*, 17 (8), 743–769.

Weibel, R. and Dutton, G., 1998. Constraint-based automated map generalization. *In*: T. Poiker and N. Chrisman (eds.) *Proc. 8th International Symposium on Spatial Data Handling (SDH'98)*, Vancouver, Canada, 11–15 July 1998. IGU, 214–224.

Wertheimer, M., 1938. Laws of organization in percetional forms. *In*: W. Ellis, ed. *A source book of Gestalt psychology*. London, UK: Routledge & Kegan Paul, 71–88.

Williams, J.C., 2002. A zero-one programming model for contiguous land acquisition. *Geographical Analysis*, 34 (4), 330–349.

Wright, J., ReVelle, C. and Cohon, J., 1983. A multiobjective integer programming model for the land acquisition problem. *Regional Science and Urban Economics*, 13, 31–53.

Yaolin, L., Molenaar, M. and Kraak, M.J., 2002. Semantic similarity evaluation model in categorical database generalization. *In*: Vol. XXXIV, Part 4 of *the International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences.* ISPRS.

Zoltners, A.A. and Sinha, P., 1983. Sales territory alignment: a review and model. *Management Science*, 29 (11), 1237–1256.

SUPPLEMENTARY ON-LINE MATERIAL FOR:

Area aggregation in map generalisation by mixed-integer programming

JAN-HENRIK HAUNERT

ALEXANDER WOLFF

Chair of Computer Science I
University of Würzburg
Am Hubland
D-97074 Würzburg

Appendix A. Approaches for solving mixed-integer programs

In this appendix we sketch the method that we used to solve our mixed-integer programs. This method is called *branch-and-cut*, which combines two different methods, namely *branch-and-bound* and *cutting plane algorithms*. We give an outline of these two methods and then briefly explain their combination.

Consider the objective function

$$\text{minimise} \qquad x_1 + 2x_2 \tag{A1}$$

$$\text{subject to the constraints} \qquad 16x_1 + 12x_2 \geq 35 \tag{A2}$$

$$16x_1 - 12x_2 \leq 17. \tag{A3}$$

In this two-dimensional LP, each constraint corresponds to a half plane; the intersection of the half planes is the set S of feasible solutions (shaded region in Figure A1(a)). Among the points in S we are interested in one that minimises the objective function. The coefficients of the variables in the objective function yield a vector c, here $c = \binom{1}{2}$. If we sweep the plane in direction c with a line ℓ orthogonal to c, then the first points of S swept by ℓ minimise (A1). The dashed lines in Figure A1(a) depict the traces of ℓ. We obtain the optimal solution $s^* = \binom{1.625}{0.75}$.

In an *integer* linear program (IP) all variables are restricted to integer values:

$$x_1, x_2 \in \quad , \tag{A4}$$

thus the point s^* becomes infeasible. The new optimal solution becomes $\bar{s} = \binom{1}{2}$.

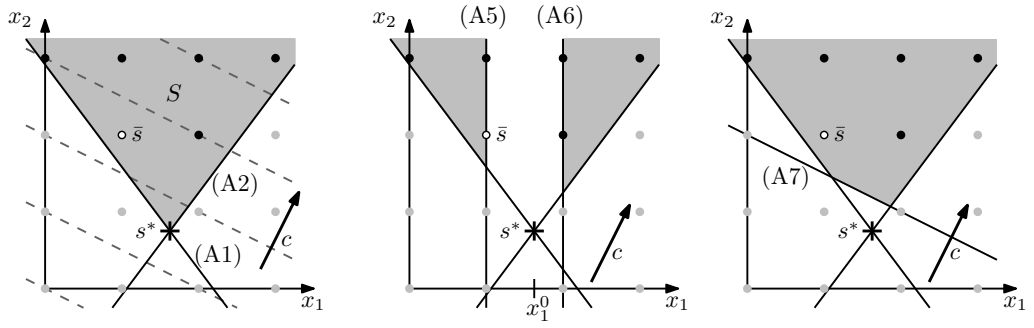
An IP can be solved with branch-and-bound and cutting plane algorithms; both start by solving the *LP relaxation* of the MIP, that is, the constraints $x_1, x_2 \in$ are relaxed to $x_1, x_2 \in$. The LP relaxation can be solved, for example, by applying the simplex algorithm. To 'branch' in branch-and-bound means to select an integer variable x_i with fractional value x_i^0 in the solution of the LP relaxation and to split the original problem into two subproblems. Each subproblem inherits all constraints of the original problem plus one of the constraints $x_i \leq \lfloor x_i^0 \rfloor$ and $x_i \geq \lceil x_i^0 \rceil$, that is, the value x_i^0 is rounded down and up, respectively. In our example we select variable x_1 (which has a fractional value of 1.625) for branching and create two subproblems, each by adding one of the constraints

$$x_1 \leq 1 \tag{A5}$$

$$x_1 \geq 2, \tag{A6}$$

as shown in Figure A1(b). The branching procedure can be applied to the subproblems, resulting in a tree where each node corresponds to a MIP. The procedure terminates if for each leaf node the solution of the LP relaxation satisfies the integrality constraints or the LP relaxation is infeasible. The best found integral solution is the optimal solution of the MIP.

In order to avoid that the tree becomes too large, we can use the fact that the LP relaxation offers a *lower bound* for the objective function, that is, the solution of the MIP cannot be better than this. Suppose that at a certain node of the branching tree an integral solution with objective value z is found, all nodes with lower bound greater or equal z can be *killed*, that is, do not need to be further investigated. The success of this method clearly depends on whether the bounds are tight or not, where tight means that the solution of the LP relaxation is a good approximation for the solution of the MIP. Unfortunately, the MIP formulations



(a) The optimal solution s^* of the LP (A1)–(A3) and the optimal solution $\bar{s}$ of the IP (A1)–(A4).

(b) branching at variable x_1 yields two subproblems

(c) the IP after addition of a cut

Figure A1.: An IP can be solved by first solving its LP relaxation (a) and then proceeding with branching (b) and cutting (c).

that appear in practise are often weak, that is, the best fractional solution is much better than the best solution of the MIP. Because of this, other methods such as cutting plane algorithms are often applied.

A *cutting plane* or simply *cut* is an additional constraint, that is, an inequality that removes the solution of the LP relaxation from the feasible region of the MIP, but does not exclude any integer solution. A cut for our example is the constraint

$$x_1 + 2x_2 \geq 4, \tag{A7}$$

which is shown in Figure A1(c). The general approach of cutting plane algorithms is to iteratively solve the LP relaxation and to add a cut, until the solution of the LP relaxation satisfies the integrality constraints. In this case, the optimal solution of the MIP is found. Different methods exist to find cuts; one of the most prominent is the algorithm of Gomory (1958). We found constraint (A7) using this approach.

Cutting plane algorithms are normally not efficient when being applied alone. Usually, the algorithms first converge to the optimal integer solution relatively fast, that is, relatively big parts of the feasible region are cut away, but later this process drastically slows down. However, by adding cuts to a MIP, its LP relaxation becomes stronger. Therefore, after adding a limited number of cuts to a MIP, the above-mentioned branch-and-bound method gets more efficient. Generally, this combination of both methods is called *branch-and-cut*. Different versions exist, for example, cuts are added to the original MIP only, or to subproblems as well (Mitchell 2002). Branch-and-cut techniques are implemented in several commercial software packages. For our experiments, we used the software ILOG CPLEX™ 11.2.

We conclude that finding a MIP formulation with a strong LP relaxation is essential for the success of most solution techniques. Of course, another factor is the size of the MIP, that is, the number of variables and constraints that are needed to encode a problem. This, however, is often considered less important.

References

Gomory, R.E., 1958. Outline of an algorithm for integer solutions to linear programs. *Bulletin of the American Mathematical Society*, 64 (5), 275–278.

Mitchell, J.E., 2002. Branch-and-cut algorithms for combinatorial optimization problems. *In*: P.M. Pardalos and M.G.C. Resende, eds. *Handbook of Applied Optimization*. Oxford, UK: Oxford University Press, 65–77.

Appendix B. A MIP for area aggregation based on network flow

In this appendix we give the complete formulation of our flow MIP, which we briefly discussed in Section 5 of our article. Recall that the flow MIP guarantees globally optimal solutions for instances of the aggregation problem. However, since we observed a very high computation time with the flow MIP, we focussed our discussion on a heuristic approach: we introduced the P-R MIP that is based on a precedence relationship. With this approach the computation time was reduced. In theory the P-R MIP does not guarantee the globally optimal solution, but in our experiments, which we documented in Section 7, it yielded near-optimal solutions. The flow MIP uses the definition of flow on the graph G. After a brief explanation of the idea, we present the formulation in detail. Figure B1 illustrates the flow model.

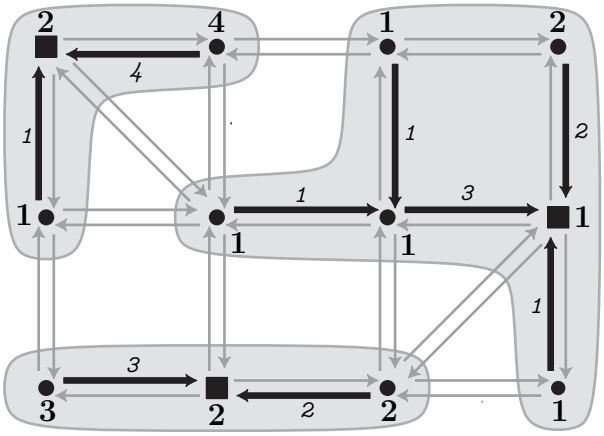


Figure B1.: Connectivity based on our flow model. Arcs with positive flow ($f_a > 0$ and $F_a = 1$) are displayed in bold black. Node weights are displayed by bold numbers, values of the flow variables f_a by italic numbers. Sinks ($s_v = 1$) are displayed by squares, sources ($s_v = 0$) by circles. All aggregates exactly satisfy the weight threshold $\theta = 7$.

Flow variables $f_a \in \ ^+$ define the amount of a commodity being transported on arcs $a \in A$, with $A = \{uv \mid \{u, v\} \in E\}$. The arcs with positive flow define a directed graph. The weakly connected components of this graph yield the aggregates in our model. To indicate whether an arc carries a positive flow, a binary variable F_a is introduced for each arc $a \in A$. The variable F_a is set to 1, if $f_a > 0$. Introducing binary variables b_{vk}, we express whether a node $v \in V$ receives colour $k \in \Gamma$. To ensure that all nodes in an aggregate receive the same colour, we enforce that two adjacent nodes should receive the same colour, if a connecting arc carries positive flow. For each node $v \in V$, a binary variable s_v is introduced, to express whether it is a sink or a source of the network. Sources contribute a positive amount of flow to the network that is equal to their weight. Thus, it is not possible to create a connected component that only contains sources, and so, fixing the colour of a sink, we are sure to have at least one node of unchanged colour in each aggregate. Aggregates of sufficient size are obtained by claiming that a sink receives a certain amount of flow: the sum of this flow and the sink's own weight must not fall below the weight threshold defined for the colour of the sink.

Let's investigate the defined requirements for the flow in the example of Figure B1: the node in the upper left corner is a sink, whose own weight is two. In order to satisfy the weight threshold of seven, the incoming flow needs to exceed the outgoing flow by at least five units. In this example, the requirement is exactly fulfilled. The node in the second row and third column of the grid-like graph is

a source. It receives a flow of two and emits a flow of three, thus its net outflow is one, which equals its weight. Since these constraints are fulfilled for all nodes, the resulting partition is feasible. The following list summarises the introduced variables:

$f_a \in [0, M]$	flow on arc $a \in A$, with $M = \sum_{v \in V} w(v)$ being the total weight of the graph.
$F_a \in \{0, 1\}$	$F_a = 1$ if arc $a \in A$ carries positive flow.
$b_{vk} \in \{0, 1\}$	$b_{vk} = 1$ if and only if node $v \in V$ receives colour $k \in \Gamma$.
$s_v \in \{0, 1\}$	$s_v = 1$ if and only if node $v \in V$ is a sink.

With the variables b_{vk}, the objective for minimum colour change is defined as follows:

$$\text{minimise} \quad \sum_{v \in V} \sum_{k \in \Gamma} w(v) \cdot d(\gamma(v), k) \cdot b_{vk} \tag{B1}$$

To link the variables F_a and f_a, such that $F_a = 1$ if $f_a > 0$ we add the following constraint:

$$M \cdot F_a \geq f_a \qquad \text{for all } a \in A \tag{B2}$$

The next constraint ensures that each node is assigned to one colour:

$$\sum_{k \in \Gamma} b_{vk} = 1 \qquad \text{for all } v \in V \tag{B3}$$

Constraint (B4) ensures that two adjacent nodes receive the same colour if a connecting arc carries positive flow.

$$b_{uk} \geq b_{vk} + (F_{uv} - 1) \qquad \text{for all } uv \in A, k \in \Gamma \tag{B4}$$

To ensure that a sink keeps its original colour, the following constraint is added:

$$b_{v,\gamma(v)} \geq s_v \qquad \text{for all } v \in V \tag{B5}$$

The next two constraints ensure the connectivity and weight-feasibility of aggregates. We will discuss their effect in detail.

$$\sum_{a=vu \in A} f_a - \sum_{a=uv \in A} f_a \geq w(v) - s_v \cdot M \qquad \text{for all } v \in V \tag{B6}$$

$$\sum_{a=vu \in A} f_a - \sum_{a=uv \in A} f_a \leq w(v) - s_v \cdot \theta(\gamma(v)) \qquad \text{for all } v \in V \tag{B7}$$

We consider first the case that node v is a source, meaning $s_v = 0$. Then, the two inequalities can be summarised by one equation:

$$\sum_{a=vu \in A} f_a - \sum_{a=uv \in A} f_a = w(v) \qquad \text{for all } v \in V$$

This means just that the net outflow from a source equals the node's weight. In the case that $s_v = 1$, meaning that v is a sink, constraint (B6) is relaxed and

constraint (B7) becomes:

$$\sum_{a=uv\in A} f_a - \sum_{a=vu\in A} f_a + w(v) \geq \theta(\gamma(v)) \qquad \text{for all } v \in V,$$

meaning that the sum of the net inflow to a sink and its own weight, is not smaller than the threshold. So, the requirement for the weight of a contiguous aggregate with equal colour is fulfilled.

Finally, we show how to optimise the compactness of aggregates according to the measure $c_{\text{shortest-path}}$, which we introduced in Section 3.3. We first add a constraint that ensures that an aggregate contains only one sink. We achieve this by insisting that, for each source (a node u with $s_u = 0$), there is at most one outgoing arc with positive flow and, for each sink (a node u with $s_u = 1$), there is no outgoing arc with positive flow::

$$\sum_{a=uv\in A} F_a \leq 1 - s_u \qquad \text{for all } u \in V. \tag{B8}$$

Consequently, the arcs with $F_a = 1$ form trees, each having a single sink. Now the flow variables f_a can be used to express the compactness measure $c_{\text{shortest-path}}$. To explain this, let's observe a flow from its source v to the sink: for each arc a of the taken path, the variable f_a increases by $w(v)$. Because of this, we can express the aim for compactness as follows:

$$\text{minimise} \qquad \sum_{a=uv\in A} \delta(u,v) \cdot f_a. \tag{B9}$$

Optimally solving the MIP with this objective, the selected tree of arcs with positive flow will automatically become equal to the aggregate's shortest path tree – otherwise the solution will not be optimal. The resulting cost for an aggregate is just the same as $c_{\text{shortest-path}}$.

Note that the objectives (B1) and (B9) can be combined in a weighted sum in order to define a trade-off.

Appendix C. A sample processed with the proposed generalisation method

In this appendix we show a larger sample that we processed with the developed generalisation method to give an impression on the effectiveness of the proposed work flow. For the same sample we show

- the original dataset of scale 1:50,000 (Figure C1),
- the preprocessed dataset, that is, the dataset after collapsing narrow polygons and polygon parts to lines (Figure C2),
- the result of applying our aggregation method to the preprocessed dataset (Figure C3), and
- the result of applying the line simplification method to the aggregated dataset (Figure C4).

The final result meets the specifications for the target scale 1:250,000. We would need to apply a smoothing and displacement of lines in order to create a visually pleasing map. This, however, is not the objective of database generalisation.

Figure C1.: A part of the dataset "Buchholz in der Nordheide" of the German topographic database ATKIS DLM 50. The whole dataset has an extent of 22 km × 22 km and contains 5461 areas. It corresponds to a map sheet of the topographic map at scale 1:50,000.

Figure C2.: The result of applying the collapse method to the sample in Figure C1. Narrow polygons and polygon parts are collapsed to lines. The two small figures show this effect for a clipping of the large figure (marked by a box).

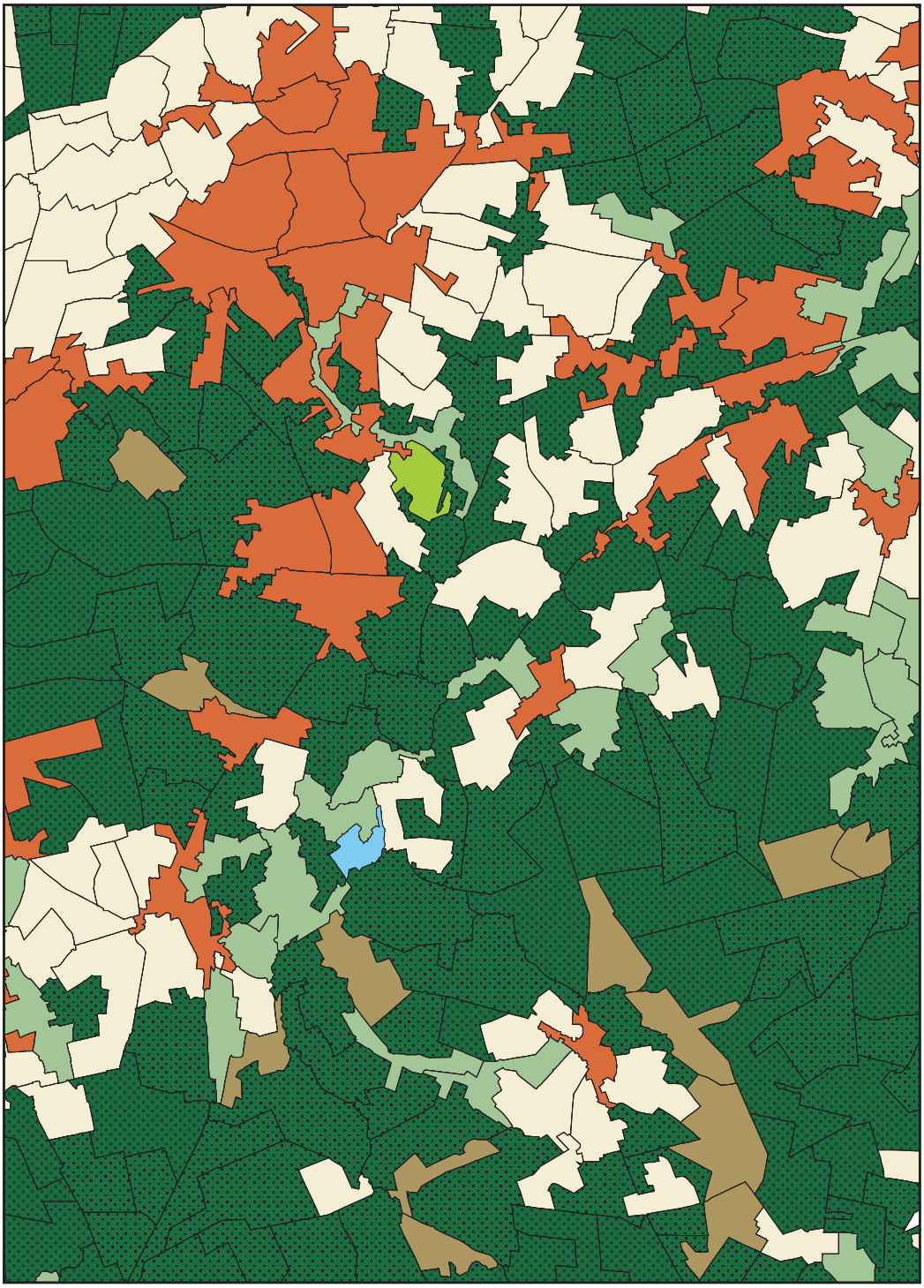


Figure C3.: The result of applying our aggregation method to the preprocessed dataset in Figure C2. The aggregation was achieved using Algorithm 2, applying $K = 200$. The P-R MIP was used in conjunction with the centre heuristic and the distance heuristic to solve the occurring problem instances. The minimal sizes were defined according to existing specifications of the German topographic database ATKIS DLM 250, which corresponds to a topographic map at scale 1:250,000. The class distances were defined according to Table 1 in our article. Both the class changes and the compactness of aggregates were considered.

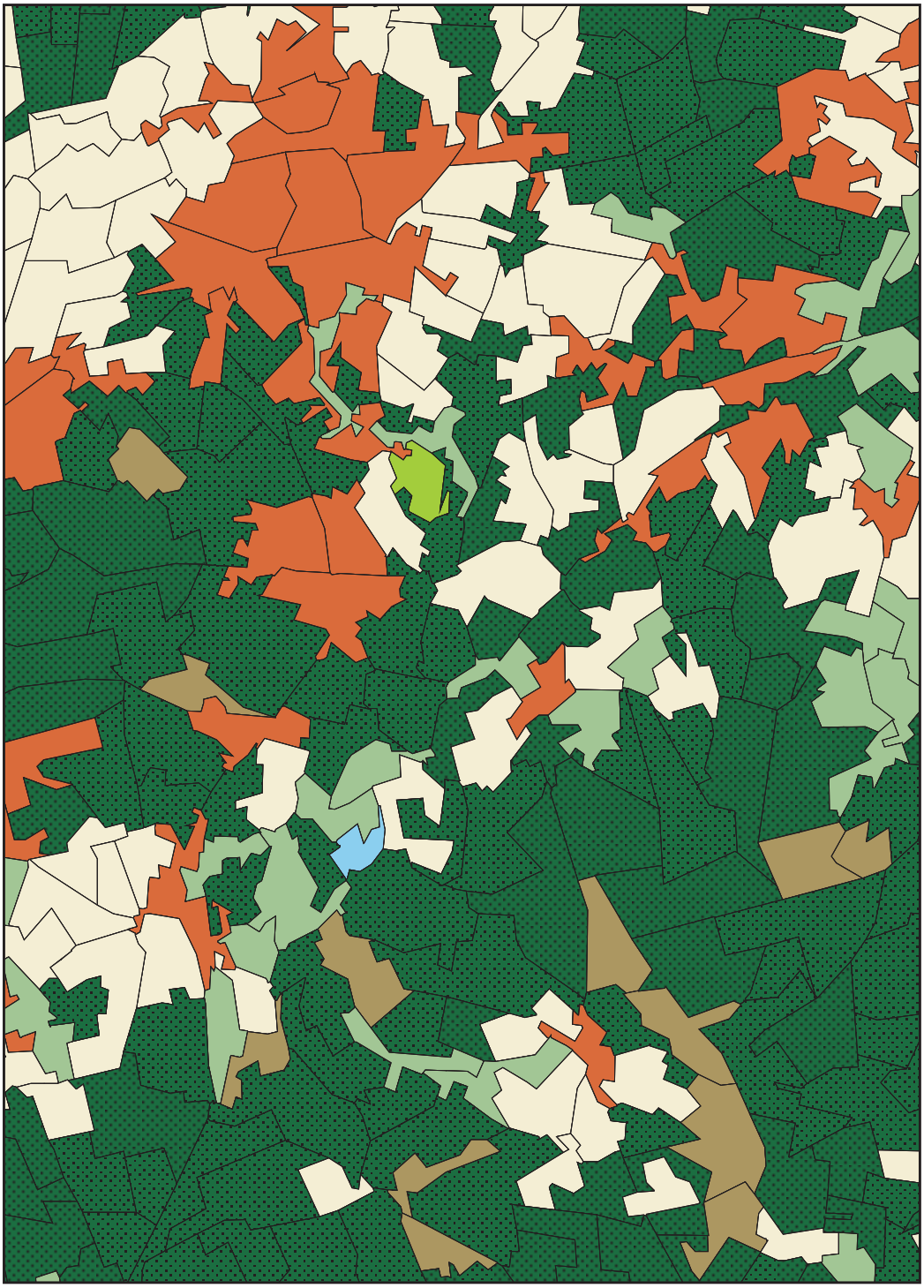


Figure C4.: The result of applying the line simplification method to the aggregated dataset in Figure C3.