Optimal and Topologically Safe Simplification of Building Footprints

Jan-Henrik Haunert
Institut für Informatik, Lehrstuhl I
Universität Würzburg
Am Hubland, 97074 Würzburg
jan.haunert@uni-wuerzburg.de

Alexander Wolff
Institut für Informatik, Lehrstuhl I
Universität Würzburg
Am Hubland, 97074 Würzburg
www1.informatik.uni-wuerzburg.de

ABSTRACT

We present an optimization approach to simplify sets of building footprints represented as polygons. We simplify each polygonal ring by selecting a subsequence of its original edges; the vertices of the simplified ring are defined by intersections of consecutive (and possibly extended) edges in the selected sequence. Our aim is to minimize the number of all output edges subject to a user-defined error tolerance. Since we earlier showed that the problem is NP-hard when requiring non-intersecting simple polygons as output, we cannot hope for an efficient, exact algorithm. Therefore, we present an efficient algorithm for a relaxed problem and an integer program (IP) that allows us to solve the original problem with existing software. Our IP is large, since it has $O(m^6)$ constraints, where m is the number of input edges. In order to keep the running time small, we first consider a subset of only $O(m)$ constraints. The choice of the constraints ensures some basic properties of the solution. Constraints that were neglected are added during optimization whenever they become violated by a new solution encountered. Using this approach we simplified a set of 144 buildings with a total of 2056 edges in 4.1 seconds on a standard desktop PC; the simplified building set contained 762 edges. During optimization, the number of constraints increased by a mere 13 %. We also show how to apply cartographic quality measures in our method and discuss their effects on examples.

Categories and Subject Descriptors

H.2.8 [**Database Management**]: Database Applications—*Spatial Databases and GIS*; I.3.5 [**Computer Graphics**]: Computational Geometry and Object Modeling

General Terms

Algorithms

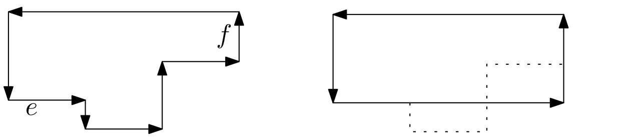
Keywords

GIS, Cartography, Map Generalization, Building Simplification, Optimization, Integer Programming

1. INTRODUCTION

Map generalization is the problem of deriving a less detailed and more abstract representation from a given geographic data set. A typical map generalization problem is the simplification of building footprints. The problem is similar to the classical line simplification problem, but, as buildings are highly regular man-made structures, special characteristics need to be considered. Therefore, solutions to both problems have been developed, on the whole, independently.

In this paper, we present an optimization approach to building simplification that is inspired by an existing approach to line simplification. Our motivation to approach the problem by optimization is, of course, to obtain generalization results of higher quality compared to other methods. There is, however, a second reason for our approach. With our method it is possible to apply various optimization objectives and constraints. We assume that comparing the results will help to better understand the criteria that make up well-generalized buildings. This will be useful for quality assessment – a problem that is considered highly relevant in the generalization literature [1, 9].



(a) original building (b) result of selecting shortcut (e, f)

Figure 1: Building simplification based on shortcuts.

Our paper builds up on work presented earlier [13]: we define the simplified outline of a building by selecting a subsequence of the original edges. New vertices are introduced at intersections of consecutive (and possibly extended) edges in the selected sequence. With this approach we keep the edge slopes fixed and so give consideration to shape regularities. For example, if the original building is rectilinear, the simplified building will automatically be rectilinear, too. An alternative view of the same problem is to define the solution with a set of shortcuts; each shortcut omits some input edges, see Fig. 1. The basic optimization problem is to minimize the number of output edges subject to a given distance tolerance. Additionally, self-intersections and intersections with other polygons must be avoided. We have shown that the problem becomes NP-hard if we forbid intersecting poly-

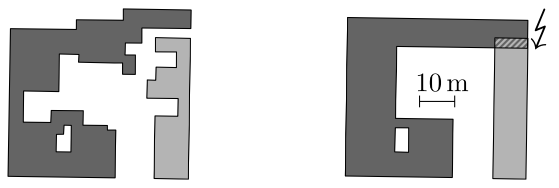


Figure 2: Two disjoint input polygons (left) and a result of the ArcGIS building simplification tool (right). The simplified polygons intersect. The simplification tolerance was set to 10 m.

gons in the output [13]. Since we cannot hope for an efficient exact algorithm we turned to integer programming. We presented experiments for single polygons without holes.

In this paper we present three innovations that render our approach applicable and our experimental results more conclusive. First, our new implementation allows us to treat sets of multiple buildings (with holes) as one optimization instance. In this setting, forbidding unwanted edge intersections becomes much more relevant. Figure 2 shows two polygons whose simplifications (which were obtained with the commercial software ArcGIS) intersect – we are now able to avoid such cases. With this extension we early encountered the limitations of our existing approach: the set-up time for the IP and the number of constraints increased drastically with the number m of input edges. Therefore, as a second innovation, we fundamentally changed the way constraints are set up and handled during optimization. Instead of explicitly modeling *all* constraints of the problem in terms of linear inequalities and applying the optimizer as a black box, we set up the IP only with a subset of the constraints that ensures some basic properties of the solution. If we choose this approach, we need to check whether a solution encountered by the optimizer is feasible and, if the solution is infeasible, we need to tell the optimizer how to continue. For this purpose, we insert so-called lazy constraints to the IP at runtime. The initial size of the IP reduces from $O(m^6)$ to $O(m)$. In our experiments, we needed to introduce only a few of the initially neglected constraints. We set up the initial IP and find the lazy constraints using a plane-sweep algorithm. As a third innovation, we show how to apply more advanced objective functions in order to improve the cartographic quality of the generalized buildings.

Our paper is structured as follows. We first review related work on line simplification and building simplification (Sect. 2). Then we give a formal definition of the problem (Sect. 3). We present our general approach using shortcuts, which yields an efficient solution to a relaxed problem (Sect. 4), our integer-programming solution (Sect. 5), and our experimental results (Sect. 6). Finally, we discuss open problems and conclude the paper (Sect. 7).

2. RELATED WORK

A classical method for building simplification [20] uses rules that are successively applied to the polygonal outline of a building. It is difficult, however, to find a good rule set and to decide for an order in which the rules are to be applied. A more generic approach [15] is to define a cell decomposition of the plane by lines approximating parts of the original building outline; some of the cells are selected to form the output polygon. Furthermore, morphological operators have been proposed for building simplification [5, 16]. Optimization (that is, least-squares adjustment) has been applied to adjust a previously simplified building ground plan [19]. The decision about which details to select, however, has not been approached by optimization yet.

In contrast to building simplification, line simplification has been frequently approached by optimization. A basic idea to simplify a line is to select a minimum subsequence of its original vertices [8]. A simplified line is often considered feasible if it satisfies the *bandwidth criterion*, that is, if the original line is within an ε-buffer of the simplified line. In order to find an optimal simplification, shortest-path algorithms can be applied to an appropriately defined graph of shortcuts. This approach allows different optimization objectives to be applied [3]. The problem becomes more involved with different objectives and constraints, but solutions have been found for the preservation of angles [4], distances [10], areas [2], and topological relations [7]. Though the optimization approach to line simplification is very powerful, a method selecting a subsequence of the original line vertices should not be applied to building simplification. For example, in order to represent the building in Fig. 1(a) as a rectangle, we need to introduce a new vertex – this is not possible with the classical vertex-based line simplification approach. With our edge-based approach, however, we can define the rectangle by selecting and extending four of the original edges.

Obviously, the generalization of building sets cannot be solved entirely by outline simplification. For example, we would also need to aggregate buildings, remove buildings, or fill holes representing small inner yards. These problems are not solved with our method but could be handled by preprocessing the data with a morphological operator [5]. After simplifying the outline, we may also apply a (least-squares) adjustment [19], for example, to make an almost rectilinear building (whose angles are close to 90 degrees) rectilinear. Instead of using our algorithm in a fixed process chain, we could use it as one basic component in a higher-level system that automatically decides which algorithm to apply in a given situation – multi-agent systems have been proposed for that purpose and indeed they have been used for building generalization [21]. We intend to consider more than a single building during simplification in order to avoid unwanted intersections. We think, however, that the instances we have to solve in practice will not become arbitrarily large but will contain about 100 buildings. This is realistic, for example, since buildings within one mesh of the road network may be generalized independently of others [18].

In an earlier work [12] we have applied integer programming to another map generalization problem, namely the aggregation of areas in land cover maps. Our success motivated us to develop a similar approach for building simplification.

3. PROBLEM DEFINITION

We now give a formal problem definition of the simplification problem. We first define a problem with the basic objective of minimizing the number of output edges (Sect. 3.1) and then introduce more advanced objective functions (Sect. 3.2).

3.1 Basic problem definition

We define the problem BUILDINGSETSIMPLIFICATION as follows:

We require a set of disjoint simple polygons with holes and the error tolerance $\varepsilon > 0$ as input. Each input polygon consists of one exterior ring (that is, a closed sequence of edges) and any number of interior rings. For an edge sequence r we denote the i-th edge by $e(r, i)$ and the number of edges by $|r|$. The edges of an exterior ring are in counterclockwise order and the edges of an interior ring are in clockwise order.

The problem is to find for each input ring r a new ring r' that satisfies the following requirements:

(R1) Each edge in r' *corresponds* to an original edge in r, which means that both edges intersect and have the same (directed) supporting line. For an edge e in r' we denote the corresponding edge by $\mathrm{corr}(e)$.

(R2) The sequence $\left(\mathrm{corr}(e(r', 1)), \mathrm{corr}(e(r', 2)), \ldots, \mathrm{corr}(e(r', |r'|))\right)$ is a subsequence of r.

(R3) Each pair (a, b) of consecutive edges in r', that is, each of the pairs $\left(e(r', i), e(r', (i+1) \mod |r'|)\right)$ with $i = 1, 2, \ldots, |r'|$, defines two polygonal chains $p'(a, b)$ and $p(a, b)$, see Fig. 3. Both $p'(a, b)$ and $p(a, b)$ begin at the last point of a that lies on $\mathrm{corr}(a)$ and end at the first point of b that lies on $\mathrm{corr}(b)$; $p'(a, b)$ is the piece of r' between these two points and $p(a, b)$ is the piece of r between these two points. We require that the Hausdorff distance between $p'(a, b)$ and $p(a, b)$ does not exceed ε.

Additionally, we require for each pair (a, b) of different and non-consecutive output edges (that may lie on the same or on different output rings):

(R4) a and b do not intersect.

We call a set of rings that satisfies requirements R1–R4 a *feasible solution*. The problem is to find a feasible solution with the minimum number of edges.

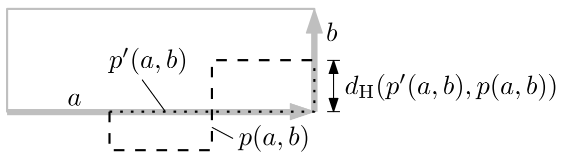


Figure 3: The polygonal chains $p'(a, b)$ and $p(a, b)$ for the edges a and b in Fig. 1. Requirement R3 says that the Hausdorff distance d_{H} between $p'(a, b)$ and $p(a, b)$ must not exceed ε.

Requirement R1 restricts us in how we can create new edges. We may only extend or shorten an old edge (at both ends) and we must not turn its direction. Requirement R2 means that we must not change the order of the edges. Intuitively, requirement R3 is similar to the bandwidth criterion that is often applied to line simplification. The original piece of a line must lie in the ε-buffer of its simplified version and vice versa. The Hausdorff distance between two point sets X and Y in $\mathbb{R}^2$ is defined as

$$d_{\mathrm{H}}(X, Y) = \max\left\{\sup_{x \in X} \inf_{y \in Y} d(x, y), \sup_{y \in Y} \inf_{x \in X} d(x, y)\right\}, \quad (1)$$

where sup is shorthand for supremum and inf for infimum. The function $d \colon \mathbb{R}^2 \to \mathbb{R}_0^+$ is the Euclidean distance. Requirement R4 ensures the simplicity of the output rings. We call the problem that remains when removing this requirement RELAXEDBUILDINGSETSIMPLIFICATION.

3.2 A more advanced objective function

In addition to the basic optimization objective of minimizing the number of edges we define three cost measures c_{area}, c_{regular}, and c_{similar}. For each pair (a, b) of consecutive edges in a solution $\mathcal{S}$ we add a term to each of the three measures.

The idea behind the first measure c_{area} is that a local simplification should have little impact on the area of a building.

$$c_{\mathrm{area}} = \sum_{(a,b) \in \mathcal{S}} \mathrm{area}(p(a, b), p'(a, b)), \quad (2)$$

where $\mathrm{area}(p(a, b), p'(a, b))$ is the amount of change in area when replacing the piece $p(a, b)$ of a ring by $p'(a, b)$.

The idea behind the second measure c_{regular} is that regular shapes and thus angles close to 90 degrees are preferred.

$$c_{\mathrm{regular}} = \sum_{(a,b) \in \mathcal{S}} \cos^2 \angle ab, \quad (3)$$

where $\angle ab$ is the angle between the two edges a and b. This implies that no cost is charged for angles of 90 degrees.

The idea behind the third measure c_{similar} is that the edge directions of the simplified building outline should be similar to the edge directions of the original building outline. For a polygonal line p we define $\mathrm{hist}(p)$ as the histogram of the edge directions in p; for all histograms we discretize the directions using the same intervals. Suppose p contains a horizontal edge of 2 m length and a vertical edge of 1 m length, then $\mathrm{hist}(p)$ contains two peaks. We define that the height of the peaks reflects the edge lengths, that is, the first peak (at 0 degree) is twice as high as the second peak (at 90 degrees), since the horizontal edge is twice as long as the vertical line. We now define

$$c_{\mathrm{similar}} = \sum_{(a,b) \in \mathcal{S}} \delta\left(\mathrm{hist}(p(a, b)), \mathrm{hist}(p'(a, b))\right), \quad (4)$$

where δ is a distance between two histograms. We use a distance that is easy to compute. For two histograms A and B with values $(a_1, a_2, \ldots, a_k)$ and $(b_1, b_2, \ldots, b_k)$, we define

$$\delta(A, B) = \sum_{i=1}^{k} |a_i - b_i|. \quad (5)$$

It remains to combine the three different measures into one objective function that is to be minimized. We combine the measures as a weighted sum, that is, we aim to minimize

$$m' + w_{\mathrm{area}} \cdot c_{\mathrm{area}} + w_{\mathrm{regular}} \cdot c_{\mathrm{regular}} + w_{\mathrm{similar}} \cdot c_{\mathrm{similar}}, \quad (6)$$

where m' is the number of output edges and w_{area}, w_{regular}, $w_{\mathrm{similar}} \in \mathbb{R}_0^+$ are weight factors that allow users to express their preferences.

4. AN APPROACH USING SHORTCUTS

In order to solve BuildingSetSimplification we select a set of *shortcuts*. A shortcut is a pair of edges in the same original ring. Selecting shortcut (e, f) means that in the solution there will be an edge a that corresponds to e, an edge b that corresponds to f, and *no* edge that corresponds to an edge between e and f; because of requirement R2, a and b will be consecutive edges in the new ring. If we want to select two consecutive input edges e and f for the simplified ring, we need to select the (trivial) shortcut (e, f).

4.1 Feasible shortcuts

Not all sets of shortcuts define feasible solutions to the problem. Moreover, selecting a single shortcut may forbid all feasible solutions. Often we can easily decide whether a shortcut (e, f) is of this type. We cannot select (e, f) if

(1) it is not possible to construct a pair (a, b) of consecutive edges that satisfy requirement R1 with $\mathrm{corr}(a) = e$ and $\mathrm{corr}(b) = f$ or
(2) each such pair violates requirement R3.

Case (1) applies if the ray $\mathrm{end}'(e)$ that starts at the first vertex of e and runs in the direction of e and the ray $\mathrm{origin}'(f)$ that starts at the second vertex of f and runs in the direction opposite to f do not intersect, see Fig. 4. To see why, we observe that the end point of an edge that corresponds to e lies on $\mathrm{end}'(e)$; the origin of an edge that corresponds to f lies on $\mathrm{origin}'(f)$. Selecting shortcut (e, f) means that both points are the same, which is not possible if $\mathrm{end}'(e)$ and $\mathrm{origin}'(f)$ do not intersect.

If case (1) does not apply, we can check whether case (2) applies by constructing the polygonal chains $p'(a, b)$ and $p(a, b)$ (with $\mathrm{corr}(a) = e$ and $\mathrm{corr}(b) = e$) and comparing their Hausdorff distance with ε. Notice that with the shortcut (e, f) we are given enough information to unambiguously determine these chains though we can neither determine the origin of the edge a corresponding to e nor the end point of the edge b corresponding to f. Figure 5 shows a shortcut that can be excluded by case (2). We call a shortcut that cannot be excluded by case (1) or (2) *feasible*.

The Hausdorff distance between two polygonal chains with m_1 and m_2 vertices can be computed in $O(m_1 m_2)$ time by computing each distance between an edge of one chain and a vertex of the other chain. For each pair of consecutive edges (a, b), the polygonal chain $p'(a, b)$ contains at most two edges. Therefore, we can compute the Hausdorff distance between $p'(a, b)$ and $p(a, b)$ in $O(m_r)$ time, where m_r is the number of edges in the ring r of $\mathrm{corr}(a)$ and $\mathrm{corr}(b)$. This implies that we can decide in $O(m_r)$ time whether a shortcut is feasible. Since there are $O(m_r^2)$ pairs of edges in r, the set of feasible shortcuts S_r of r contains $O(m_r^2)$ elements and we can compute S_r in $O(m_r^3)$ time.

In order to reduce the set-up time for S_r, we can avoid testing all pairs of edges of r for feasible shortcuts. Our idea is to use the ε-buffer β_r of r, that is, the union of all radius-ε disks centered on the edges of r. Because of requirement R3, each feasible simplification r' of r lies in β_r. Therefore, we can maximally extend the edges of r until they touch the boundary of β_r. We only need to test, for each intersection of two maximally extended edges, whether the two involved input edges form a feasible shortcut. A piecewise linear approximation of the ε-buffer β_r (which suffices our purpose) is found in $O(m_r)$ time with a Minkowski-sum algorithm; the maximal extensions of the edges and their intersections are found in $O(m_r \log m_r + k \log m_r)$ time with a plane-sweep algorithm, where k is the number of intersections [6]. This approach is favorable since ε is usually small; in this case, one can assume that k is linear in m_r.

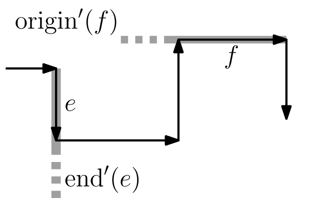


Figure 4: Shortcut (e, f) is infeasible, since $\mathrm{end}'(e)$ and $\mathrm{origin}'(f)$ do not intersect.

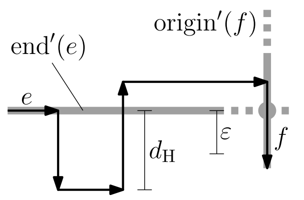


Figure 5: Selecting shortcut (e, f) violates requirement R3; therefore, (e, f) is infeasible.

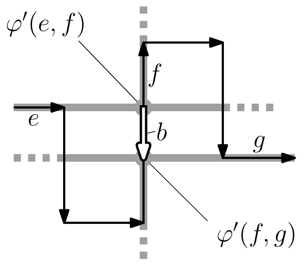


Figure 6: Shortcuts (e, f) and (f, g) are feasible but cannot be selected simultaneously, since edge b from $\varphi'(e, f)$ to $\varphi'(f, g)$ doesn't have the same direction as f.

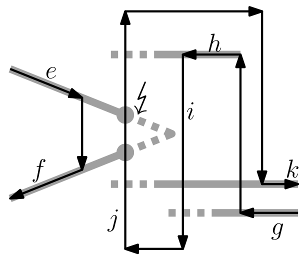


Figure 7: Selecting shortcut (e, f) alone violates requirement R4 (see $\lightning$), but (e, f) can be selected together with (h, j) or with (g, i) and (j, k).

4.2 Feasible combinations of shortcuts

Next we treat combinations of feasible shortcuts that yield feasible solutions to BuildingSetSimplification. This directly leads to an efficient algorithm for RelaxedBuildingSetSimplification and will later allow us to express the more constrained problem as an IP.

For each input ring r we introduce a directed graph $G_r = (S_r, A_r)$ whose node set is the set of feasible shortcuts. The arc set A_r contains an arc between two (consecutive) shortcuts (e, f) and (f, g) in S_r if and only if the edge b that begins at the intersection point $\varphi'(e, f) = \mathrm{end}'(e) \cap \mathrm{origin}'(f)$ and ends at the point $\varphi'(f, g) = \mathrm{end}'(f) \cap \mathrm{origin}'(g)$ has the same direction as f. Figure 6 shows that there may be pairs of consecutive and feasible shortcuts that do not satisfy this condition. Since each arc in A_r is defined by three edges in r, A_r has $O(m_r^3)$ elements. The set A_r can be found in $O(m_r^3)$ time since testing each potential element for the right direction takes constant time.

Selecting a pair of consecutive shortcuts (e, f) and (f, g) determines whether the edge $b = (\varphi'(e, f), \varphi'(f, g))$ with $\mathrm{corr}(b) = f$ becomes an edge of the output ring. The whole output ring is thus determined by a closed sequence of consecutive shortcuts, that is, a cycle in G_r. Our idea is to search for a shortest cycle C_r in G_r. Selecting the shortcuts of such a cycle yields an optimally simplified ring.

Theorem. *The shortcuts corresponding to the set* $\{C_r \mid r \text{ input ring}\}$ *form an optimal solution for* RelaxedBuildingSetSimplification. *The set* $\{C_r\}$ *can be computed in* $O(m^5)$ *time.*

Proof. Each simplification r' of a ring r that satisfies requirements R1–R3 corresponds to a cycle in G_r: for each triplet of consecutive edges (a, b, c) in r', the node set S_r contains the two shortcuts $(\mathrm{corr}(a), \mathrm{corr}(b))$ and $(\mathrm{corr}(b), \mathrm{corr}(c))$, and the arc set A_r contains an arc connecting the two. Conversely, for each cycle in G_r, there is a simplification r' satisfying requirements R1–R3: for each arc $((e, f), (f, g))$ of the cycle, the ring r' contains the edge $(\varphi'(e, f), \varphi'(f, g))$. The number of edges of an output ring r' is equal to the number of shortcuts in the corresponding cycle since for each shortcut (e, f) there is a vertex in r' (that is, the point $\varphi'(e, f)$). Therefore, a cycle with the minimum number of nodes yields a solution with the minimum number of edges. Finding a shortest cycle in an unweighted digraph $G = (V, E)$ takes $O(|V| \cdot |E|)$ time [14]. Since $|S_r| \in O(m^2)$ and $|A_r| \in O(m^3)$, we can solve RelaxedBuildingSetSimplification in $O(m^5)$ time. $\square$

We can solve the relaxed problem with the advanced objective (6) by associating each graph node s (that is, each shortcut) with a weight $\omega(s)$. For the shortcut (e, f) and the edges a and b corresponding to e and f we define the weight to be 1 *plus* the weighted sum of the three terms charged for (a, b) in equations (2)–(4). We can now solve the problem by finding a minimum-weight cycle in G_r, since the weights of the selected shortcuts sum up to objective (6). Since the nodes in G_r are topologically sorted, we can find the cycle of minimum weight through a given node in $O(|S_r| + |A_r|)$ time by dynamic programming. Doing this for all nodes, the overall running time becomes $O(|S_r| \cdot |A_r|)$, that is, we obtain the same bound as in the unweighted case.

We now have a polynomial-time solution to the relaxed problem, but it is far too slow and we still don't have a solution to the more constrained original problem. Indeed, the simplifications that we find may be infeasible according to requirement R4. For example, selecting the shortcut (e, f) in Figure 7 yields two self-intersections. It seems that we could fix this simply by excluding shortcuts such as (e, f) from the set S_r. This conclusion, however, is false. In the example, we would obtain a feasible solution by selecting (e, f) together with the shortcut (h, j). Alternatively, we could very well select (e, f) together with the shortcuts (g, i) and (j, k).

In our earlier work [13] we have shown that simplifying buildings with a minimum number of output edges is NP-hard if we forbid self-intersecting polygons as output. This was our justification for turning to integer programming. In the next section we apply the same technique but we handle constraints in a fundamentally different way.

5. INTEGER PROGRAMMING

Integer programming is a method for combinatorial optimization, which we have earlier reviewed [11]. It generally means to model a problem as an integer program (Sect. 5.1) and to solve it with an optimizer (Sect. 5.2).

5.1 IP formulation

Our integer program (IP) for BuildingSetSimplification has the binary variables

$$x_s \in \{0, 1\} \quad \text{for each } s \in S_r \text{ and each } r \in R$$

with $x_s = 1$ if and only if shortcut s is selected for the simplified building set.

Since the sum of the weights associated with the selected shortcuts equals objective (6), our aim is to

$$\text{minimize} \quad \sum_{r \in R} \sum_{s \in S_r} \omega(s) \cdot x_s \,, \tag{7}$$

where R is the input set of polygonal rings.

We enforce with two constraints that the union of the shortcuts $s \in S_r$ with $x_s = 1$ define a cycle in the graph G_r.

Constraint (8) ensures that for each edge $e(r, j)$ in ring r there is exactly one shortcut omitting $e(r, j)$ or starting at $e(r, j)$.

$$\sum_{s \in S_r^j} x_s = 1 \quad \text{for each } e(r, j) \text{ and } r \in R\,, \tag{8}$$

with $S_r^j = \big\{(e(r, i), e(r, k)) \in S_r \mid j \in \{i\,, i+1\,, \ldots\,, k-1\}\big\}$. Note that in the definition of S_r^j the range for j is cyclic, that is, if $k < i$, then $j \in \{i\,, i+1\,, \ldots\,, n\,, 1\,, \ldots\,, k-1\}$.

Constraint (9) forbids that two consecutive shortcuts (e, f) and (f, g) are selected together if this implies a change of direction for the edge corresponding to f (recall Fig. 6).

$$\begin{aligned} &x_s + x_t \leq 1 \quad \text{for each } s = (e, f), t = (f, g) \in r \text{ and } r \in R \\ &\quad \text{where edges } (\varphi'(s), \varphi'(t)) \text{ and } f \text{ have different directions.} \end{aligned} \tag{9}$$

It remains to forbid intersecting edges. Recall that each potential output edge is determined with a pair of consecutive shortcuts. We can avoid a potential intersection in the output by forbidding the corresponding pair of intersecting output edges, which is determined with four shortcuts (two pairs of consecutive shortcuts). We must not select all four of them. This is expressed with the following constraint.

$$\begin{aligned} &x_s + x_t + x_u + x_v \leq 3 \\ &\quad \text{for each } s = (e, f), t = (f, g) \in r_1 \text{ and } r_1 \in R \\ &\quad \text{and each } u = (h, i), v = (i, j) \in r_2 \text{ and } r_2 \in R \\ &\quad \text{where edges } (\varphi'(s), \varphi'(t)) \text{ and } (\varphi'(u), \varphi'(v)) \text{ intersect} \\ &\quad \text{but edges } f \text{ and } i \text{ do not intersect.} \end{aligned} \tag{10}$$

5.2 IP solution

The number of instances of constraint (10) is $O(m^6)$ but may be small in practice. For example, if ε is smaller than half the distance between two buildings, intersections between the two buildings will not occur. Nevertheless, setting up this constraint requires substantial time for analyses; testing each two of the $O(m^3)$ potential output edges for intersection is not practicable. Similarly, constructing each potential output edge and testing it for the right direction is time-consuming. Therefore, we neglect constraints (9) and (10) when setting up the IP, that is, we start the optimization *only with constraint* (8), which has $O(m)$ instances. The time for the IP set up becomes dominated by the search for feasible shortcuts. Obviously, if we solved the reduced IP with a standard algorithm, we would possibly obtain infeasible solutions. We need to tailor the algorithm in order to take care of the neglected constraints.

We solve the problem with the optimization software IBM ILOG CPLEX 12.1.0, which uses a branch-and-cut algorithm (refer to Mitchell [17] for the basic principles). This software offers programming interfaces that allow users to implement so-called callbacks. Callbacks are invoked at certain states of the branch-and-cut algorithm; by implementing a callback, a user can take control over the optimization process. We use an *IncumbentCallback*, which is invoked whenever the branch-and-cut algorithm encounters a new incumbent solution (that is, a solution to the IP that is better than any other solution so far). We test whether the incumbent solution contains edges of wrong direction and intersecting edges (again, we use the plane-sweep algorithm [6] for this task). This allows us to check whether the incumbent solution is feasible according to the neglected constraints (9) and (10) without explicitly formulating them. If the incumbent solution is infeasible, we reject it and set up the violated constraints, which we temporarily store in a list. In addition to the IncumbentCallback we use a *LazyConstraintCallback*. Whenever this is invoked, the constraints are removed from the list and inserted to the IP as *lazy* constraints. The idea behind lazy constraints is that the LPs that are solved when solving the IP can be kept smaller when these constraints are not included. CPLEX will, however, include a lazy constraint in the LP as soon as it becomes violated. In other words, the solution computed by CPLEX makes sure that all the lazy constraints that have been added are satisfied. Additionally, we need a *BranchCallback* in order to ensure that there is always a node where the algorithm can continue branching. For more technical details on callbacks we refer to the CPLEX[1] user's manual for .

6. EXPERIMENTAL RESULTS

We implemented our method in Java using the ILOG CPLEX callable library for integer programming. Additionally, we used the plane-sweep algorithm of the Java Topology Suite[2] (JTS) to find segment intersections.

We tested our method for building footprints of the metropolitan Boston area, using the computer specified in Table 1. The data set is freely available as part of the Massachusetts Geographic Information System, MassGIS[3]. According to the data specifications, the building footprints were manually extracted from LiDAR data. Table 1 summarizes our experimental results for a sample of this data set, see Fig. 8.

[1]`http://public.dhe.ibm.com/software/websphere/ilog/docs/optimization/cplex/ps_usrmancplex.pdf`
[2]`http://www.vividsolutions.com/jts/jtshome.htm`
[3]`http://www.mass.gov/mgis/lidarbuildingfp2d.htm`

weight setting	W1		W2	
ε	10 m	20 m	10 m	20 m
# variables	7040	8422	7040	8422
# initial constraints	2056	2056	2056	2056
# lazy constraints	245	690	91	272
# output edges	923	708	939	762
$0.01 \cdot c_{\text{area}}[\text{m}^2]$	168.52	358.51	78.52	103.71
$1.00 \cdot c_{\text{regular}}$	49.25	45.27	43.46	37.16
$0.01 \cdot c_{\text{similar}}$	81.42	156.46	61.32	92.49
overall cost	923.00	708.00	1122.30	995.37
IP set-up time	0.86 s	1.13 s	0.89 s	1.14 s
IP solution time	2.47 s	6.59 s	1.17 s	3.00 s
overall time	3.33 s	7.72 s	2.06 s	4.14 s

Table 1: Results of our experiments with the test data set in Fig. 8, which contains 2056 edges. Computation times are in seconds CPU time. All experiments were performed on a Windows PC with 3 GB RAM and a 3.00 GHz Intel dual-core CPU.

This sample covers Boston's North End and contains 144 buildings and 2056 edges. We processed this sample with four different parameter settings. The second and third column of Table 1 summarize the results of minimizing the number of output edges, that is, we applied objective (6) and the weight setting

$$\text{(W1)}\quad w_{\text{area}} = w_{\text{regular}} = w_{\text{similar}} = 0\,.$$

The fourth and fifth column of Table 1 summarize the results with objective (6) using the weight setting

$$\text{(W2)}\quad w_{\text{area}} = 0.01\,\frac{1}{\text{m}^2}\,, w_{\text{regular}} = 1\,,\ \text{and}\ w_{\text{similar}} = 0.01\,.$$

With both settings we tested the effect of different error tolerances, that is, we set $\varepsilon = 10\,\text{m}$ and $\varepsilon = 20\,\text{m}$. Figure 9 shows the output with the setting W1 and $\varepsilon = 10\,\text{m}$. Figures 10 and 11 show the outputs with the setting W2 using the error tolerances $\varepsilon = 10\,\text{m}$ and $\varepsilon = 20\,\text{m}$, respectively.

According to Table 1, the number of variables increases by 20 % with the higher error tolerance. This is because more shortcuts become feasible. The number of variables, however, stays within the same magnitude as the number of edges. By allowing for more variation, violations of the neglected constraints become more likely, thus more of the lazy constraints need to be added during optimization. Both the increase in the number of variables and the increase in the number of lazy constraints result in a longer IP solution time. It is interesting, however, that the IP solution time (though theoretically exponential in the number of variables) stays within the same magnitude as the time for setting up the IP. The longest running time, which we encountered with W1 and $\varepsilon = 20\,\text{m}$, was 8 seconds. For this instance, our simplification algorithm reduced the number of edges by 63 %.

Comparing the results for both weight settings, we observe that the number of lazy constraints decreases if we use our combined objective, that is, the setting W2. With W1, the size of the set of constraints increases by at most 690 (34 % of the original size); with W2, however, only 272 constraints

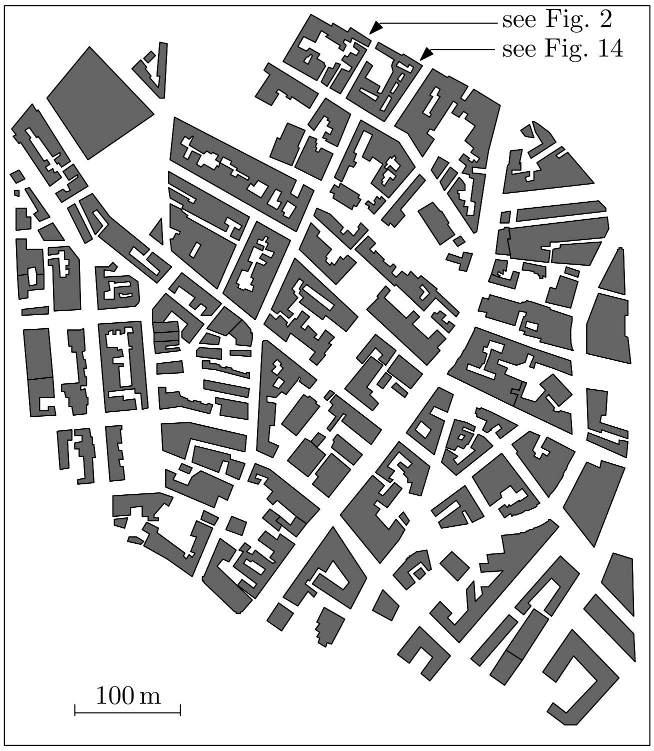


Figure 8: Input dataset of North End, Boston with 144 polygons and 2056 vertices.

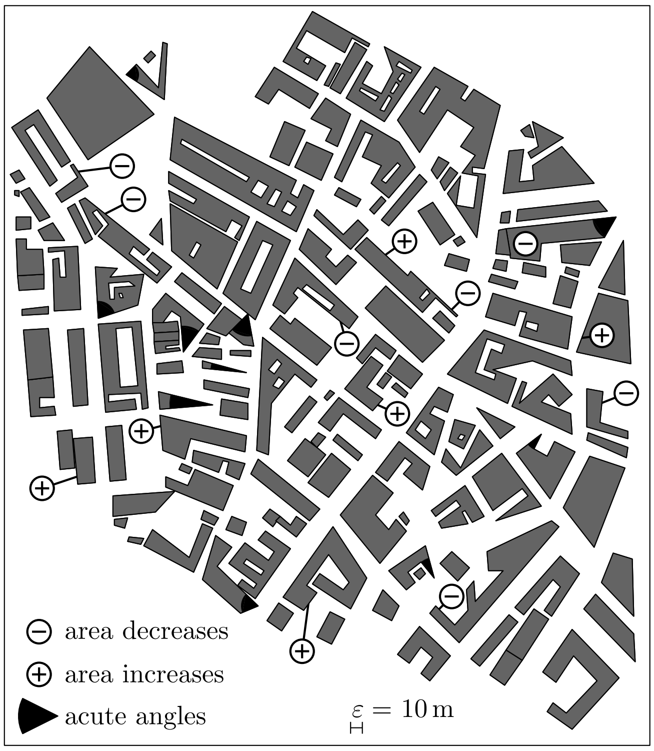


Figure 9: Result of our algorithm minimizing the number of vertices subject to the error tolerance $\varepsilon = 10\,\mathrm{m}$.

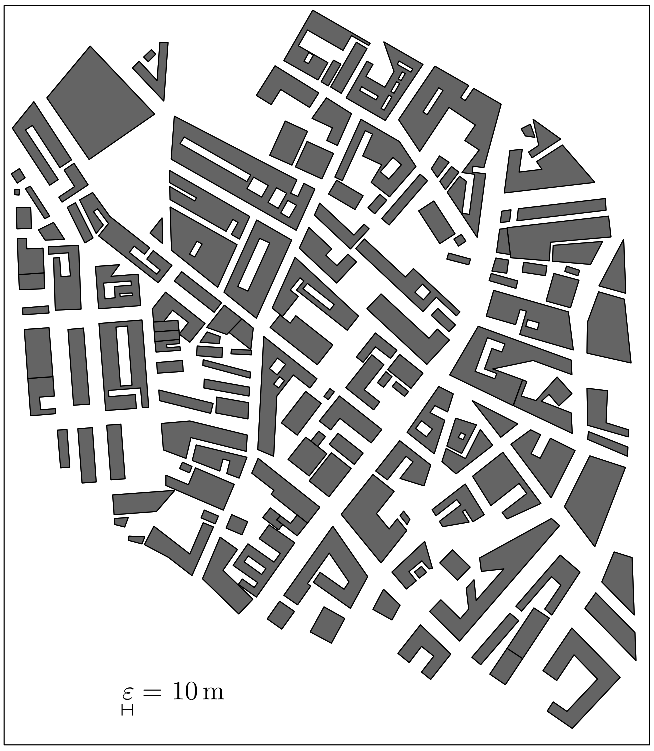


Figure 10: Result of our algorithm minimizing the combined cost function with the weight setting W2 subject to the error tolerance $\varepsilon = 10\,\mathrm{m}$.

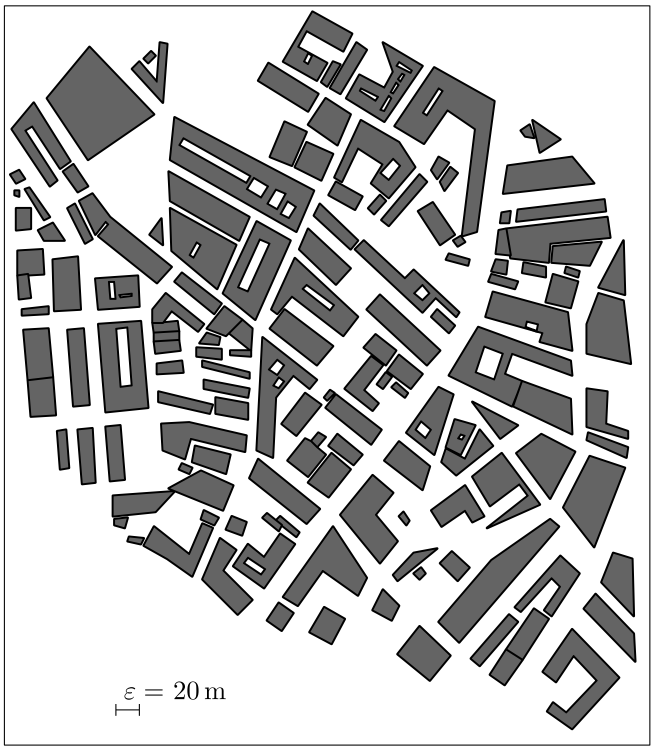
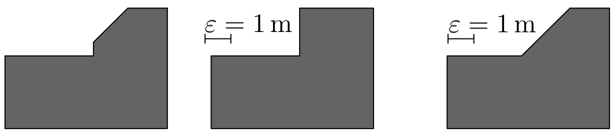


Figure 11: Result of our algorithm minimizing the combined cost function with the weight setting W2 subject to the error tolerance $\varepsilon = 20\,\mathrm{m}$.



(a) input (b) output (regular) (c) output (similar)

Figure 12: An input building (a) with two simplifications. The output building becomes rectilinear when considering c_{regular} (b) but more similar to the input when considering c_{similar} (c).

are added (13 %). Possibly, this is because many solutions violating the neglected constraints are expensive according to the combined objective and thus become pruned during optimization. As a consequence, the IP solution time decreases ($-53\,\%$ for $\varepsilon = 10\,\mathrm{m}$ and $-54\,\%$ for $\varepsilon = 20\,\mathrm{m}$).

Obviously, with the setting W2, the three measures c_{area}, c_{regular}, and c_{similar} (which, with W1, do not contribute to the overall cost) decrease at the expense of additional output edges. It is interesting, however, that a few additional edges suffice to greatly decrease c_{area} and c_{similar}. With $\varepsilon = 10\,\mathrm{m}$, the number of edges increases by a mere 2 % but the sum of our three additional cost measures decreases by 39 %. Figures 9 and 10 reveal that these changes indeed reflect quality improvements. In Fig. 9 we labeled some simplified buildings with "+" and "−", which reflects increases and decreases in area. Comparing these results with the results for the same buildings in Fig. 10 that were obtained with the setting W2 we observe that, by considering the measure c_{area}, the differences to the original footprints become visibly less.

By applying the measure c_{regular} right angles become preferred and by applying the measure c_{similar} the dominating edge directions become preserved. For our data set, both measures have similar effects, since most of the input buildings are almost rectilinear. Without considering both measures we obtain many acute angles that were not dominant in the input (black filled slices in Fig. 9) but with the weight setting W2 we obtain buildings that better reflect our objectives (Fig. 10). The measures c_{regular} and c_{similar}, however, may be opposed to each other. For example, Fig. 12(a) shows a building that, for $\varepsilon = 1\,\mathrm{m}$, has two feasible simplifications with six edges, see Figures 12(b) and 12(c). We obtained the simplification in Fig. 12(b) with the weight setting $w_{\mathrm{area}} = w_{\mathrm{similar}} = 0$ and $w_{\mathrm{regular}} = 1$, that is, we only considered one of our three quality measurs: the objective of generating right angles. Indeed, all six output vertices have right angles. In contrast, the simplification in Fig. 12(b) was obtained with the weight setting $w_{\mathrm{area}} = w_{\mathrm{regular}} = 0$ and $w_{\mathrm{similar}} = 0.01$, that is, we only considered the similarity of the edge directions. Indeed, the relatively long diagonal input edge becomes selected for the output. The difference between c_{regular} and c_{similar} is thus that c_{regular} is in a way progressive (it pushes the building towards being more regular) while c_{similar} is conservative (it avoids the building becoming dissimilar to the input).

Our experiments show that our method avoids unwanted edge intersections, also in difficult situations. Figure 13 shows a building footprint of a church ruin whose roof has been destroyed. The polygon thus contains one exterior ring and one interior ring – only one wall thickness separates both. If we neglect constraint (10), both rings intersect after simplification (Fig. 13(b)). By considering constraint (10), however, we obtain two disjoint output rings (Figures 13(c)–(h)). Due to the difficulty of avoiding edge intersections when simplifying the polygon, we needed to add maximally 96 lazy constraints during optimization; the size of the set of constraints increased by 93 %. This result was attained for the output in Fig. 13(f). However, the running time was still low: the processing took 0.17 seconds (on the machine used for the experiments in Table 1). Again, our experiments show that applying the setting W1 does not yield well-generalized buildings (Figures 13(c)–(d)). The main directions of the building edges are not preserved and, for $\varepsilon = 8\,\mathrm{m}$, both rings change drastically in size. With our combination of different objectives, however, we obtain better generalization results (Figures 13(e)–(h)). Perhaps, a human cartographer could improve on the result in Fig. 13(f) by preserving the symmetry of the building.

7. FUTURE WORK AND CONCLUSION

In this section we discuss two possibilities to further improve on our method. First, our current method does not preserve containment relationships between different polygonal rings (Sect. 7.1). Second, we would like to preserve symmetries (Sect. 7.2). We conclude our paper in Sect. 7.3.

7.1 Preserving containment relationships

Requirement R4 forbids unwanted edge intersections. The containment relationships of different polygonal rings, however, may change. We may add the following requirement:

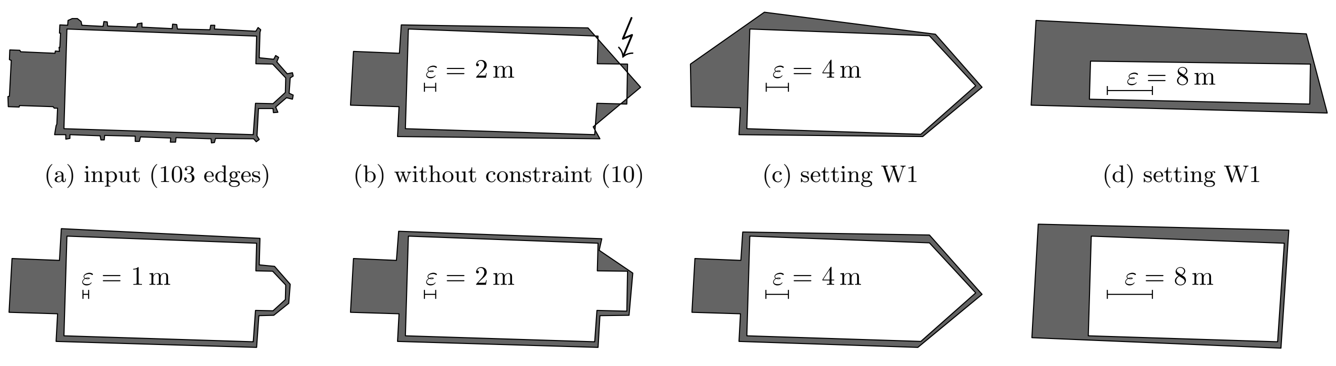
(R5) In the output, a polygonal ring is completely contained in a second polygonal ring if and only if the same relationships holds for the corresponding input rings.

Figure 14 shows an example from our Boston dataset that violates this requirement but satisfies requirements R1–R4. Two interior rings become exterior. Similarly, if we apply our algorithm to two disjoint polygons, we may obtain two polygons of which one completely contains the second one. In the example in Fig. 14 we could simply forbid the shortcut (e, f) in order to preserve the containment relationships. In general, however, it does not suffice to consider single shortcuts or even quadruples of shortcuts (which allowed us to express requirement R4 as constraint (10)). Figure 15 shows an example that is more involved. Whether applying the shortcut (e, f) removes the hole from the polygon's interior depends on how the hole is simplified: requirement R5 may be violated (Fig. 15(b)) or satisfied (Fig. 15(c)).

Nevertheless, we could apply our approach of using callbacks: if we detect two rings that violate requirement R5, we need to add a lazy constraint. Let A and B be the sets of shortcuts that are selected to form the two rings. We may add the constraint

$$\sum_{a \in A} x_a + \sum_{b \in B} x_b \leq |A| + |B| - 1 \qquad (11)$$

which simply rules out the solution. Unfortunately, the number of such constraints may become exponential in the number of variables. In order to find out whether this approach

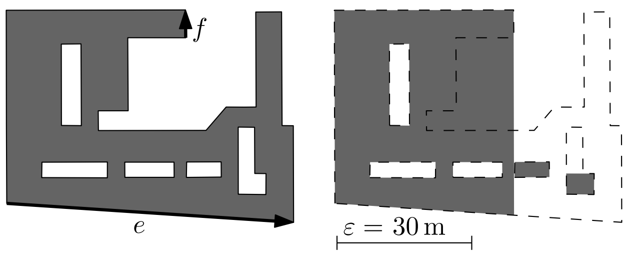


$\varepsilon = 2\,\mathrm{m}$
$\varepsilon = 4\,\mathrm{m}$
$\varepsilon = 8\,\mathrm{m}$
$\varepsilon = 1\,\mathrm{m}$
$\varepsilon = 2\,\mathrm{m}$
$\varepsilon = 4\,\mathrm{m}$
$\varepsilon = 8\,\mathrm{m}$

(a) input (103 edges) (b) without constraint (10) (c) setting W1 (d) setting W1

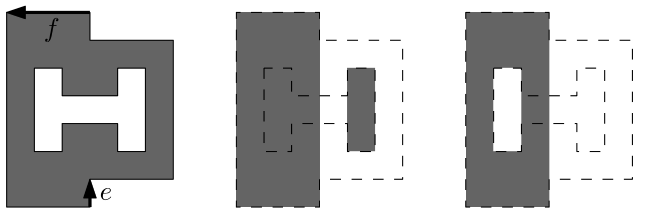
(e) setting W2 (f) setting W2 (g) setting W2 (h) setting W2

Figure 13: A footprint of a church ruin (a), a result of our algorithm with the weight setting W2 but without constraint (10) that forbids intersecting edges (b), two results with the setting W1 (c)–(d), and four results with the setting W2 (e)–(h). The results (c)–(d) as well as (e)–(h) differ because of different error tolerances.



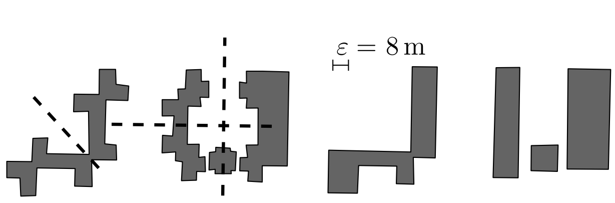
(a) input (b) output (setting W1)

Figure 14: Selecting the shortcut (e, f) implies that two interior rings become exterior.



(a) input (b) requirement R5 violated (c) requirement R5 satisfied

Figure 15: When selecting the shortcut (e, f) the interior ring may become exterior (b) or not (c) – it depends on how the interior ring is simplified.



(a) input (b) output (setting W2)

Figure 16: A set of buildings (a) with manually detected axes of symmetry (dashed lines) and a result of our algorithm (b). The symmetries are lost.

is successful in practice, we need to conduct additional experiments. We think, however, that requirement R5 will seldom become violated. For example, if we preprocess our data with a morphological closing operator [5], small holes will become eliminated. The error tolerance ε will usually be much smaller than the diameters of the remaining rings. In our experiments we observed violations of requirement R5 only when setting $\varepsilon \geq 20\,\mathrm{m}$ while using the setting W1.

7.2 Preserving symmetries

Many buildings are symmetric. Since generalization aims at preserving shape characteristics, the simplified buildings should have the same axes of symmetry as the input buildings. This criterion, however, is currently not considered in our method. Symmetries may become lost, see Fig. 16. Symmetry preservation may be a hard constraint or an optimization objective. We think that it should be modeled as an optimization objective since we may want to destroy a symmetry in order to better satisfy other objectives. For example, in order to satisfy the objective of generating right angles and the objective of minimizing the number of edges, we may want to simplify a regular octagon (which has eight axes of symmetry) into a square (which has only four axes of symmetry). For symmetry preservation we first need an algorithm that detects axes of symmetry. Then, for each pair $(u = (e, f), v = (g, h))$ of shortcuts, we could decide whether the edges e and h as well as the edges f and g are mutually mirror images. In this case, we would like to select either both shortcuts u and v or none of them – selecting only one of them would destroy the symmetry. In our IP, we could take symmetry into account by charging a cost proportional to the absolute difference of the variables x_u and x_v.

7.3 Conclusion

We have presented a new optimization approach to building footprint simplification. Our basic idea is to select a subsequence of the original edges of an input polygon; the intersections of the selected and possibly extended edges form the vertices of the simplified polygon. Our method guarantees a solution within a geometric error tolerance ε specified by the user and allows us to generate optimal solutions with respect to different objectives. As a basic optimization objective we minimize the number of output edges. We have shown, however, that we need to consider additional objectives in order to produce good generalization results. By considering measures based on size, edge directions, and angles, we were able to improve the results. This improvement requires just a few additional edges: in an experiment with $\varepsilon = 10\,\text{m}$, the number of output edges increased by a mere 2 % but the sum of our three additional cost measures decreased by 39 %. Furthermore, we have discussed the constraint of avoiding unwanted edge intersections in detail and we have presented an IP-based method that handles this constraint. In our integer-programming approach we start the optimization with a small set of constraints that ensures some basic properties of a solution. Some constraints are neglected in the beginning – these are inserted to the IP as soon as they become violated. The initial size of the IP reduces from $O(m^6)$ to $O(m)$. Our approach guarantees a globally optimal feasible solution and allows us to solve relatively large problem instances fast. For example, we simplified a set of 144 buildings with a total of 2056 edges in 4.1 seconds on a standard desktop PC; the simplified building set contained 762 edges. During optimization, the size of the set of constraints increased by a mere 13 %.

We conclude that we can simplify sets of building footprints optimally with respect to meaningful measures and that our method is fast enough to solve real-world instances. We plan to extend our method in order to preserve containment relationships and shape characteristics, for example, symmetry.

References

[1] S. Bard. Quality assessment of cartographic generalization. *Trans. GIS*, 8(1):63–81, 2004.

[2] P. Bose, S. Cabello, O. Cheong, J. Gudmundsson, M. van Krefeld, and B. Speckmann. Area-preserving approximations of polygonal paths. *J. Discrete Algorithms*, 4:554–556, 2006.

[3] G. M. Campbell and R. G. Cromley. Optimal simplification of cartographic lines using shortest-path formulations. *J. Oper. Res. Soc.*, 42(9):793–802, 1991.

[4] D. Chen, O. Daescu, J. Hershberger, P. Kogge, N. Mi, and J. Snoeyink. Polygonal path simplification with angle constraints. *Comp. Geom. Theor. Appl.*, 32(3):173–187, 2005.

[5] J. Damen, M. van Kreveld, and B. Spaan. High quality building generalization by extending the morphological operators. In *Proc. 11th ICA Workshop Generalisation and Multiple Representation*, 2008.

[6] M. de Berg, O. Cheong, M. van Kreveld, and M. Overmars. *Computational Geometry: Algorithms and Applications.* Springer, Berlin, Germany, 2008.

[7] M. de Berg, M. van Kreveld, and S. Schirra. Topologically correct subdivision simplification using the bandwidth criterion. *Cartography & GIS*, 25(4):243–257, 1998.

[8] T. Deveau. Reducing the number of points in a plane curve representation. In *Proc. Auto-Carto VII*, pages 152–160, 1985.

[9] Y. Filippovska, V. Walter, and D. Fritsch. Quality evaluation of generalization algorithms. In *Proc. XXIst ISPRS Congress*, volume 37 (part B2) of *Internat. Archives Photogrammetry, Remote Sensing and Spatial Inform. Sci.*, pages 799–804, 2008.

[10] J. Gudmundsson, G. Narasimhan, and M. Smid. Distance-preserving approximations of polygonal paths. *Comp. Geom. Theor. Appl.*, 36(3):183–196, 2007.

[11] J.-H. Haunert. *Aggregation in Map Generalization by Combinatorial Optimization.* Dissertation, Leibniz Universität Hannover, Germany, 2009.

[12] J.-H. Haunert and A. Wolff. Generalization of land cover maps by mixed integer programming. In *Proc. 14th Annu. ACM Sympos. Advances in Geograph. Inform. Syst. (ACM GIS'06)*, pages 75–82, 2006.

[13] J.-H. Haunert and A. Wolff. Optimal simplification of building ground plans. In *Proc. XXIst ISPRS Congress*, volume 37 (part B2) of *Internat. Archives Photogrammetry, Remote Sensing and Spatial Inform. Sci.*, pages 373–378, 2008.

[14] A. Itai and M. Rodeh. Finding a minimum circuit in a graph. *SIAM J. Comput.*, 7(4):413–423, 1978.

[15] M. Kada and F. Luo. Generalisation of building ground plans using half-spaces. In *Proc. ISPRS Commission IV Sympos. Geospatial Databases for Sustainable Development*, volume 36 (part 4) of *Internat. Archives Photogrammetry, Remote Sensing and Spatial Inform. Sci.*, 2006.

[16] H. Mayer. Model-generalization of building outlines based on scale-spaces and scale-space events. In *Proc. ISPRS Commission III Sympos. Object Recognition and Scene Classification from Multispatial and Multisensor Pixels*, volume 37 (part 3) of *Internat. Archives Photogrammetry, Remote Sensing and Spatial Inform. Sci.*, pages 530–536, 1998.

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

[18] N. Regnauld. Contextual building typification in automated map generalization. *Algorithmica*, 30(2):312–333, 2001.

[19] M. Sester. Optimization approaches for generalization and data abstraction. *Int. J. Geogr. Inf. Sci.*, 19(8–9):871–897, 2005.

[20] W. Staufenbiel. *Zur Automation der Generalisierung topographischer Karten mit besonderer Berücksichtigung großmaßstäbiger Gebäudedarstellungen.* Dissertation, Technische Universität Hannover, Germany, 1973.

[21] S. Steiniger, P. Taillandier, and R. Weibel. Utilising urban context recognition and machine learning to improve the generalisation of buildings. *Int. J. Geogr. Inf. Sci.*, 24(2):253–282, 2010.