Bachelor Thesis

From Many User-Contributed Polygons to One Polygon Consensus

Fabian Feitsch

Date of Submission:	June 21, 2016
Advisors:	Prof. Dr. Alexander Wolff
	Benedikt Budig, M. Sc.
	Dr. Thomas van Dijk

Julius-Maximilians-Universität Würzburg
Lehrstuhl für Informatik I
Algorithmen, Komplexität und wissensbasierte Systeme

Abstract

Given a finite set of polygons we investigated the question how to find a polygon that is as similar as possible to all of the input polygons. We call such a polygon a polygon consensus. This can be done by using a geometrical or a cluster-based approach. We propose three algorithms using these approaches. Two cluster-based algorithms were evaluated with real world data. These algorithms involve two pre-processing steps, each of them needs heuristic parameters. We could show that one of those pre-processig steps can be safely skipped without loss of quality. The decision which pre-processing step to use is based on the data of which the polygon consensus has to found. The quality of the calculated polygon consensuses is rated with a semi-automatic method also described in the thesis.

Zusammenfassung

Um Gebäude aus handschriftlichen Karten zu digitalisieren benötigt man ein Verfahren, welches aus einer Menge von Polygonen ein Polygon findet, das mit den meisten Eingabepolygonen übereinstimmt. Dies kann durch geometrie-basierte oder durch cluster-basierte Verfahren erreicht werden. Wir schlagen drei Algorithmen vor, die diese Verfahren nutzen. Zwei cluster-basierte Algorithmen evaluierten wir mit realen Daten. Diese Algorithmen benötigen zwei vorbereitetende Schritte, jeder dieser Schritte erfordert die Angabe heuristischer Parameter. Wir konnten zeigen, dass einer dieser Schritte weggelassen werden kann, ohne die Qualität der Ausgabe zu beeinträchtigen. Die Wahl, welcher der Schritte durchgeführt wird, erfolgt anhand der Art der vorliegenden Daten. Die Qualität der erzeugten Polygone wurde mit einer halbautomatischen Methode bewertet, die ebenfalls in dieser Arbeit vorgestellt wird.

Contents

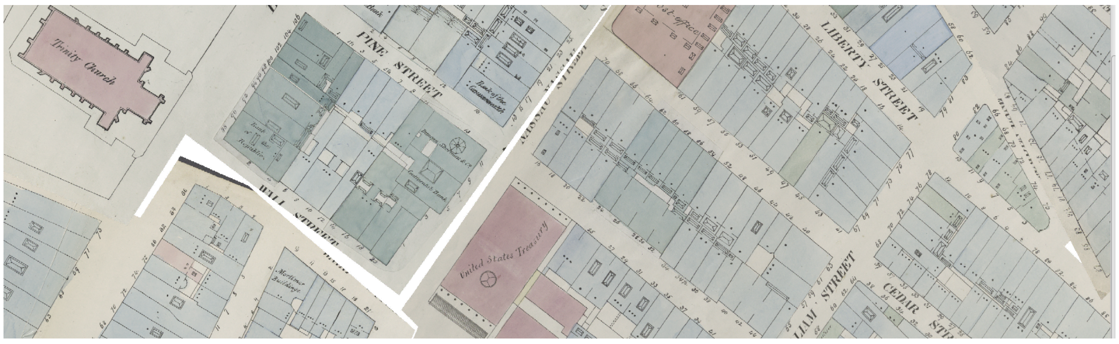


Fig. 1.1: A map of the financial district of New York in the 19th century.

1 Introduction

One benefit that came with the Internet is the ability to allow a large amount of independent users to work together on one project. This is known as crowd sourcing. A specific crowd sourced project is maintained by the New York Public Library (NYPL). The project is called *Building Inspector* and aims to create a fully digitized map of New York City in the 19th century [2]. With this map, it will be possible to explore the development of the city of New York. Which buildings were torn down to make room for the modern skyscrapers? Are there any buildings whose shapes did not change at all over the time? On which foundation was the main branch of the New York Public Library built? The amount of interesting questions is virtually infinite. The project is based on handwritten atlases of insurance companies of that time like the map in Figure 1.1. How can the structures in those atlases be digitized automatically? According to the lead developer of the *Building Inspector*, Mauricio Arteaga, *manually* digitizing one building including the recognition of several attributes like background color and address needs several minutes of time [4].

The first step of the digitization was to run an algorithm on the images. This algorithm produces polygons that approximately represent the buildings. By definition of the atlases, a building is an area which is completely enclosed by dark lines and there must be no gaps in those lines. Further, an enclosed area must have a background color other then white to be a building [4]. However, the algorithm sometimes fails and, for example, misses corners of buildings or extracts handwritings and other artifacts like folded edges. Such cases are shown in Figures 1.2a and 1.2b.

This is where the crowd becomes involved. First, people should classify the output of the algorithm as “Yes”, “Fix” or “No”. The results which received a majority of “Yes”-votes are saved instantly, the ones that received a majority of “No”-votes are deleted. Then the users are asked to correct the remaining polygons the algorithm produced. To

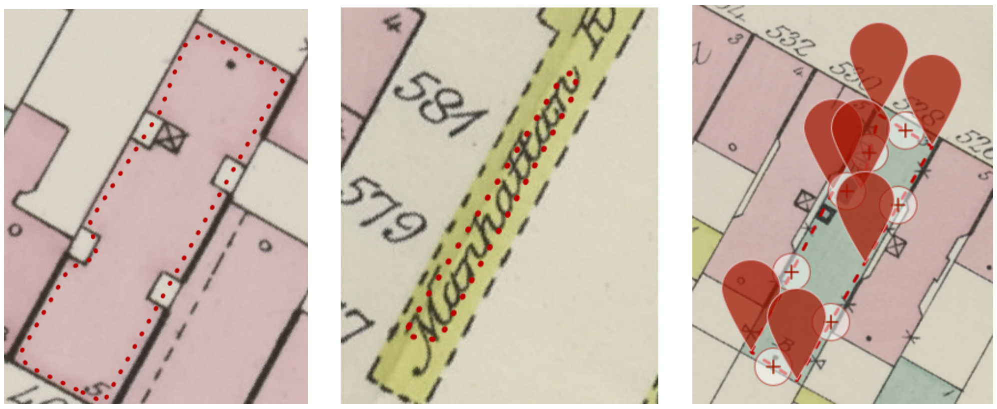

(a) This extracted polygon needs a fix of the details. **(b)** Here, a handwriting was captured, which is not correct. **(c)** The interface to fix the polygons.

Fig. 1.2: Figures 1.2a and 1.2b present two examples where the image extraction algorithm failed. Figure 1.2c depicts the interface to correct the polygons.

do this, the users have the possibility to add or remove corners and to move them freely. The interface is depicted in Figure 1.2c. Because the interface is easy to understand, web-based and responsive, it is accessible to everyone having some spare seconds.

Every single digitized building is reviewed by several persons, so that each building has several corrections proposed by users. It is very likely that most persons agree that one specific corner is part of the building. However, it is not very likely, that those people hit the same pixel in the vicinity of that corner. Further, there may be some people who think a bay window is part of the building while other users do not go into such a level of detail. The central question of this thesis is: How to construct a polygon that agrees with most of the given polygons? Namely, what is the polygon consensus based on a list of similar polygons?

The other attributes of a building, like the background color, the purpose of the building and its address are also obtained by crowd-sourcing in a similar way. However, this thesis is only concerned with the extraction of a polygon consensus representing the layout of a certain building.

In the next chapter we present some related work which is used in this thesis. The third chapter raises the question what the term “agrees with most of the given polygons” really means and defines the problem in a formal way. It also discusses which aspects of the general definition are useful for the *Building Inspector*. In chapter four and five, we present three different algorithms to calculate a polygon consensus in general. Practical results can be found in chapter six. We run the algorithms with real world data given by the NYPL and analyzed the polygon consensuses. At the end of the thesis, the reader finds proposals for further work and a summary of our findings.

2 Related Work

There are two papers whose results are used throughout this thesis. The first one regards the process of obtaining the original polygons using the historical maps. The second paper provides a solution for an important task in finding the polygon consensus of polygons whose corners lie roughly in the same areas.

2.1 Historical Map Polygon and Feature Extractor

The first step of digitizing the insurance maps is the automated extraction of the buildings. Maurico Arteaga developed a tool chain which implements this process [4]. It involves the use of several open source tools like OpenCV, R (statistical programming language), GDAL and GIMP. An insurance atlas consists of several sheets. These sheets are represented as image files. For each sheet in one atlas, the following steps are executed:

Tresholding Using GIMP, the map is greyscaled. All lines become black while the rest is colored white. This allows for easy polygon extraction in the next step.

Rough Polygon Extraction GDAL offers a function to extract polygons from an image. These polygons are filtered so that only reasonable polygons remain. Then the polygons are simplified using R. The simplification included the removal of holes and reducing the number of edges.

Polygon Exclusion and Feature Extraction Now, the polygons are compared to the initial map and further attributes are extracted, for example artifacts lying inside the polygons and their background color. For this step, GDAL is used again, together with the Python library OpenCV. All uncolored polygons are removed because the specification of the atlases defines as building as a colored shape.

According to Arteaga this process works well on the New York maps, but may have to be adopted to other maps. Several of the produced polygons are wrong, but Arteaga states that using crowd sourcing to correct the outputs of the process is still faster than the manual extraction. His algorithm to find a polygon consensus from the user-contributed polygons is described and reviewed in Chapter 5.1.

2.2 The DBSCAN-Algorithm

When n users select one corner of a building, they will probably hit n different pixels. One major task in finding the polygon consensus involves grouping those points to find the corner that was meant by the users. We call such a collection of near points which in fact mean the same corner a *cluster*. The DBSCAN-Algorithm calculates such clusters [7]. The advantage of the DBSCAN-Algorithm is the small amount of parameters needed to receive a useful result. Other popular cluster algorithms, like the CLARANS-Algorithm, need to know the amount of clusters *before* the computation starts. But in our application, we do not know how much clusters there are.

In principle, the algorithm generates a graph $G = (V, E)$. The set of vertices V is given by all points of all polygons that were provided by the users. There is an edge between two points p_1 and p_2 if the Euclidean distance between p_1 and p_2 is lower than a given min-eps. Generally speaking, a cluster is one connected component in G. Another parameter is called min-pts. If min-pts $= k$, then the value of $k-1$ describes the minimal degree at least one vertex in a connected component must have so that the component is a cluster. All connected components that do not fulfill this condition are considered as noise. Figure 2.1 shows an example of the DBSCAN-Algorithm using the depicted value for min-eps. The value of min-pts is 2. We can conclude that most of the users entered a triangle. Two users may have seen a fourth corner and selected a square, but these points are not in any cluster, because there are to few of them in that area. Probably a mistake caused the outlier at the bottom, which is ignored.

The simplest implementation of the DBSCAN-Algorithm has a running time of $\mathcal{O}(n^2)$. For one corner, the algorithm must calculate the distance to each other corner which takes n steps. In the worst case this must be done for each of the n corners, yielding a cubic running time. The algorithm could be accelerated to $\mathcal{O}(n \log n)$ by using advanced datastructures. We figured out, that the cubic running time of the simple implementation was fast enough for our data, so we used it for the sake of simplicity.

Whenever the verb "to cluster" occurs in this thesis, we used the DBSCAN-Algorithm as described above. Each process of clustering some points includes two parameters of which we silently assume we have adequate values. Of course, some thoughts on choosing those values are provided in later chapters, too.

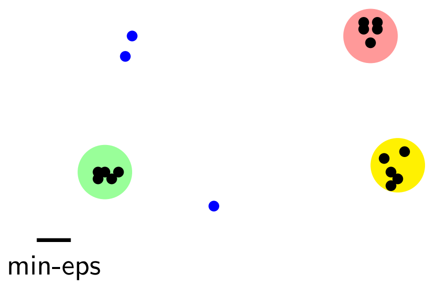


Fig. 2.1: DBSCAN applied to this set of points identifies three clusters and three noise points.

3 Preliminaries & Problem Definition

In Chapters 1 and 2 we used the term “polygon consensus” without specifying what the word “consensus” really means. In this chapter we will give a definition of the problem as basis for the following chapters. In addition, we discuss these definitions regarding the *Building Inspector*.

3.1 Basic Terms

We will use the following basic definitions throughout this thesis.

Definition 3.1. A *corner* is a 2-tuple which gives the x-coordinate and the y-coordinate of the corner. Therefore, a corner is denoted as $p = (x, y)$ with $x, y \in \mathbb{Q}$.

If we connect several corners, we get a geometrical structure. One of them is the polygon, which we define now.

Definition 3.2. A *polygon* is a n-tuple with $n > 2$ distinct corners. The corners must be ordered so that no line segment between two successive points intersect. There is also an edge between the last corner and the first corner.

For example, the polygon $p = ((1, 1), (3, 1), (2, 2))$ is a isosceles triangle. We observe that by this definition, the amount of corners in the polygon is the same as the amount of edges. Further, this definition does not allow holes in the polygons and the polygon is always a finite area.

Definition 3.3. Let P represent the infinite set of all polygons. Then P^+ contains all finite subsets of P without the empty set. An element $G \in P^+$ is called a *polygon group*, or just *group* if the context is clear.

Normally, a polygon group contains all polygons given by users for one specific building on the map of the *Building Inspector*. However, it is possible that a polygon group actually describes more buildings because it is not always clear which building the original image processing algorithm extracted. Such a polygon group is depicted in Figure 3.1. Usually most of the polygons in the group are concentrated in one area and only some polygons are scattered around. We denote such scattered polygons by *outliers* or *remote polygons* of the group.

Definition 3.4. Let f be a function with $f : P^+ \to P$. The function therefore accepts a polygon group as input and returns *one* single polygon as output. We call this polygon the *consensual polygon*.

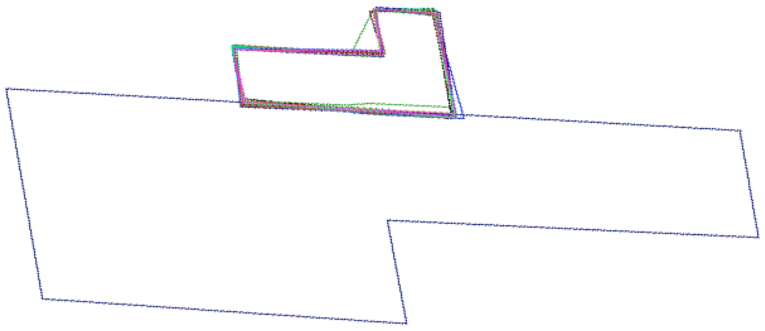


Fig. 3.1: This group describes two different buildings. The southern polygon would be considered as outlier because the majority of the users encircled the northern shape.

The *polygon consensus* is the polygon that one would consider intuitively and naturally to be the representative of a polygon group $G \in P^+$. The consensual polygon on the other side is the result of one particular algorithm calculating f on a group $G \in P^+$. The consensual polygon of a hypothetical, perfect algorithm is always the polygon consensus.

The definition of the function f is not complete yet as we did not define how the output is formed. For example, the function which returns the first polygon found in a given polygon group complies with the definition above. But we agree that the polygon consensus of that function is in general not what we would call the polygon consensus. We will explore the structural properties of a polygon consensus in the next section.

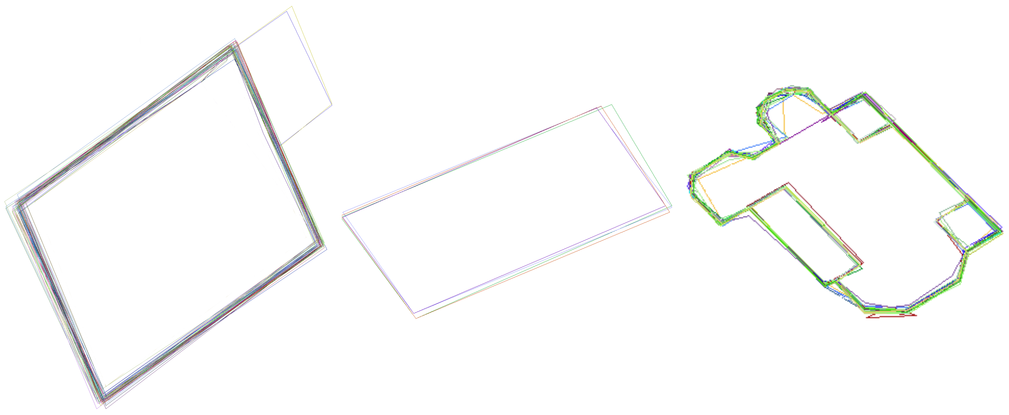


Fig. 3.2: Some example data from the NYPL. The question is *what* is the polygon consensus for each of these polygon groups. Of course, these groups are based on map images, but we will try to find the polygon consensus without the underlying pictures.

3.2 The Polygon Consensus

There are different ways to characterize the polygon consensus of a group $G \in P^+$. Look at the three polygon groups in Figure 3.2. The figures show polygon groups from the *Building Inspector*. How would you mark the polygon consensus in each of these figures? In the first picture, a large amount of polygons can be seen which are roughly the same. We would choose arbitrarily one polygon with four corners because most polygons in the group have the same four corners. However, this approach turns out to not as good as it sounds when we look at the second figure. All polygons are similar, but the corners are spread over a wider range, making each polygon distinguishable from any other polygon. In this situation, taking only one of those polygons will not work well. The most intuitive way to solve this problem is to find the centroid of each corner area and build a polygon consensus of those centroids. This approach works also in the first picture. But this approach is not applicable in the last picture because not all polygons have the same amount of corners.

We conclude from the approaches above that an algorithm which computes f should have the following properties: First, if the group G consists of many similar shaped polygons and the amount of other polygons is negligible, then the algorithm should return a polygon consensus which is similar to the majority of the polygons in G. Second, if the corners are spread widely, but the shapes of the polygons are still easily recognizable as similar polygons, the corners of the polygon consensus should be some kind of centroid of these areas. The message of the third picture is that the algorithm must identify more structural properties of the input group than the amount of corners per polygon.

As we will see, some algorithms also need additional parameters besides the group G to work properly. The effects of those parameters are often complicated to understand and best values are found empirically. We want to find algorithms without any further parameters because they are easier to apply. Even if it seems not unreasonable to adjust the parameters for calculating the polygon consensus of several thousand groups of the *Building Inspector*, we are interested in algorithms applicable in general without long calibrations in advance.

There are two categories of algorithms. The first category contains geometry-based algorithms. Those algorithms do *not* need to cluster the corners first because they are using merely geometric calculations. The second category includes algorithms which *do* cluster the corners first. After the clustering, the geometry of the polygons is not considered any more. Instead, the relations between the clusters are explored. From the first category, we will propose the Pile-Algorithm in the next chapter. Then, in Chapter 5, two cluster-based algorithms are described.

Unfortunately, we still lack a formal definition of the "polygon consensus". Every attempt to formulate a precise definition contains the the words "similar", "as much as possible" or phrases alike which we had to define in detail. We will see in the next chapter that intuitive algorithms based on the ideas above will give reasonable polygon consensuses. Therefore, we can omit giving a precise definition of the polygon consensus.

4 A Geometry-Based Algorithm

The principle of the Pile-Algorithm is natural and easy to understand. Given a polygon group $G \in P^+$ without remote polygons, we consider this group as an arbitrarily ordered pile of polygons. Now we see that the polygons overlap in some areas and new shapes on the inside are formed. These shapes are actually polygons, too. However, we denote them *faces* to distinguish them from the given polygons. Let F be the set of all faces in the pile. An attribute r is assigned to each of the faces. The number $r_s \in \mathbb{N}$ represents how many polygons overlap in the respective face s. Then the pile algorithm implements the following function $f \colon P^+ \times \mathbb{Q} \to P$:

$$f((p_1, p_2, \ldots, p_n), \alpha) = \bigcup_{\alpha \cdot n \leq i \leq n, i \in \mathbb{N}} \{s \in F \mid r_s \geq i\}$$

The union $\cup$ of several faces $F' \subseteq F$ is defined to be the polygon that encloses all faces in F' and nothing more. The parameter $\alpha \in \mathbb{Q}$ describes the level of agreement in the output polygon. An example showing the functional principle of f can be found in Figure 4.1. If $\alpha = 0$, then the polygon consensus is equivalent to the union of all polygons in the group. If $\alpha = 1$ then the polygon consensus represents the intersection of the polygon group. Values lower than 0 or greater than 1 are compatible with our definition. In the former case, the function behaves just the same as if α would be 0. In the latter case, no polygon consensus could be found. There are several existing algorithms to obtain the union of two polygons, one of them was proposed by Avraham Margalit [10]. However, with his algorithm, we cannot determine the attributes r_s efficiently. Therefore, we used another approach to find the unions.

Before we describe the algorithm itself, we have to restrict the input. Consider two polygons, one has the shape of an "E", the other one the shape of an "I", as depicted in Figure 4.2a. If we put the "I" ontop of the right side of the "E", we get a pile of polygons

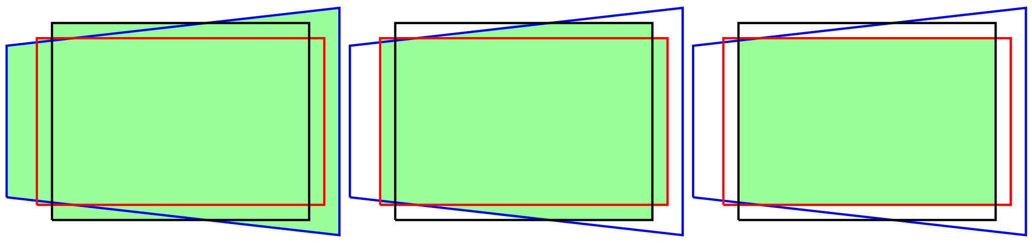

(a) $\alpha = 1/3$ **(b)** $\alpha = 2/3$ **(c)** $\alpha = 1$

Fig. 4.1: The polygon consensus with the Pile-Algorithm – with different choices for α.

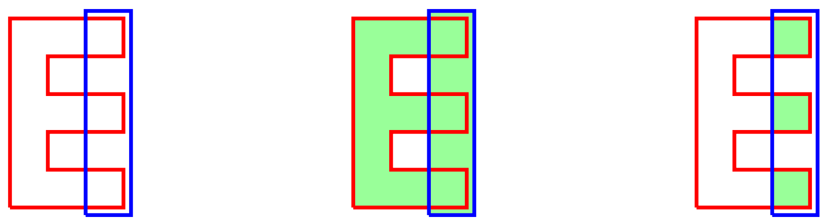

(a) The input **(b)** $\alpha = 1/2$ **(c)** $\alpha = 1$

Fig. 4.2: If we take an E-shaped polygon and an I-shaped polygon, the results of the Pile Algorithm are no polygons according to our definition.

with two holes. For $\alpha = 0$, we obtain an object which is no polygon with respect to our definition of polygons (Definition 3.2). This is shown in Figure 4.2b. It gets even worse if $\alpha = 1$. Like Figure 4.2c shows, we get three independent polygons. This contradicts our function f to calculate a polygon consensus (Definition 3.4). Therefore, we restrict the input of the Pile Algorithm so that no subset of the input group encloses any holes and every polygon overlaps with every other polygon. One might say that this restriction is too severe, but on the real life data of the *Building Inspector*, it has little effect.

The algorithm itself consists of two stages. First, a graph $\mathcal{G} = (V, E)$ is built. All corners in the input group plus all intersection points of the edges of the polygons represent the set of vertices V. In the edge set E, all edges of the polygons are gathered, but some are split with respect to the intersection points in V. Therefore $\mathcal{G}$ is planar because all intersections are replaced by vertices. The pseudocode which builds that graph is shown in Algorithm 1.

The graph is initialized with one polygon from the input group G in line 1 to 3. Then we add one polygon p at a time to the graph in the for-loop starting in line 4. Because the polygons are likely to overlap, we must not only add the corners of the polygon to our graph, but also the intersection points. These are calculated in the nested for-loops by checking every edge of p with every edge in the existing graph. If they are intersecting each other, we first split the existing edge into two new edges, divided by the intersection point. Once we have checked every existing edge, we can add the segments of the new edge to our graph. Because we know all intersections, the segments can easily be calculated and inserted in lines 24 to 25.

To check whether two edges intersect, we use basic two-dimensional geometry. We represent the edges as infinite lines. Then we obtain a linear system with two equations which we solve as if we were crossing two lines. If the solution exists, all we have to do is to check the calculated scalar variables that describe where the intersection on each of the lines can be found. If, and only if these two scalars are lower or equal to 1 while being greater or equal to 0, the intersection lies on both of the edges. This approach can yield edges of size 0 if the start or end point of an edge merely touches another edge. Such invalid edges are discarded immediately.

For clearness, one detail is omitted in the pseudocode below: Everytime we insert or replace an edge, we look if this edge is already in our set. In this case we increment the weight of this edge, given by $w(edge)$. At the end, the weight-function w holds how often each edge is present in the input group of polygons.

Algorithm 1: BuildGraph-Procedure for the Pile-Algorithm

```
Input: A group $G \in P^+$ of polygons
Output: A planar graph representing the polygons in $G$
1  Let $p_s$ be an arbitrary polygon in $G$
2  $V = \{(x, y) \mid (x, y) \in p_s\}$
3  $E$ = edges representing $p_s$ and weight of those edges is 1
4  foreach $p \in G \setminus \{p_s\}$ do
5      foreach $edge \in p$ do
6          $intersections = \emptyset$
7          foreach $oldEdge \in E$ do
8              $\{start, end\} = oldEdge$
9              if $edge$ and $oldEdge$ not parallel and intersect then
10                 $(x, y) = \text{intersectionOf}(edge, oldEdge)$
11                 $intersections = intersections \cup \{(x, y)\}$
12                 if $oldEdge$ does not start and does not end with $(x, y)$ then
13                     Replace $oldEdge$ in $E$ with the two new segments, update $w$
14             if $edge$ and $oldEdge$ parallel and overlap then
15                 $point_{\text{edge}}$ = suitable end point of $oldEdge$
16                 $intersections = intersections \cup \{point_{\text{edge}}\}$
17                 Replace $oldEdge$ in $E$ with the two new segments, update $w$
18             if $edge$ and $oldEdge$ parallel and one enclosed in other then
19                 if $edge$ longer than $oldEdge$ then
20                     $intersections = intersections \cup \{start, end\}$
21                 if $edge$ shorter than $oldEdge$ then
22                     Replace $oldEdge$ in $E$ with the three new segments, update $w$
23         $V = V \cup \{(x, y) \mid (x, y) \in p\} \cup intersections$
24         Sort $intersections$ by their distance to one end-point on $edge$
25         Insert $edge$ by adding the segments to $E$ segment by segment, update $w$
26 return $(V, E, w)$
```

With the graph generated from Algorithm 1 we can now implement the function f. Let G be the polygon group and $\mathcal{G} = (V, E)$ the graph constructed from G. Our algorithm to find the consensus with respect to α is based on the following theorem:

Theorem 4.1. *Let $0 < \alpha \cdot n \leq 1$ with $n = |G|$ and $\alpha \in \mathbb{Q}$. The polygon p_α encloses all faces in $\mathcal{G}$. By deleting all edges of $\mathcal{G}$ which are borders of p_α and have a weight of 1, we obtain a new graph $\mathcal{G}'$. Then r_t for all faces t in $\mathcal{G}'$ is at least 2.*

Proof. We must show that by deleting the edges in p_α with weight 1, we remove all those faces s with $r_s = 1$. Let e be such an edge with $w(e) = 1$. By deleting e, the face s is resolved. Since $e \in p_\alpha$ we know that s is residing in the outer regions of the graph. For r_s to be greater than 1 for such faces, the weight of all edges separating the face from the outside must be also greater than 1. But as we delete only edges with weight exactly 1, we remove only faces which are covered by one polygon. Because all faces s with r_s lie in other regions, we resolve all of them, leaving only faces covered by at least two polygons. $\square$

By taking the union of Graph $\mathcal{G}'$, the polygon consensus with $1 < \alpha \cdot n \leq 2$ can be found. To satisfy even higher values of α, we repeat this step. But before we find the surrounding polygon of $\mathcal{G}'$, we must decrease the weight of all edges in p' which were not deleted. Also, we remove the eventually created vertices with no neighbours. The details can be found in Algorithm 2.

Algorithm 2: The Pile-Algorithm

```
Input: A group of $n$ simply overlapping polygons $G$ and a parameter $0 \leq \alpha \leq 1$
Output: A polygon consisting of all areas in $G$ that are overlapped by at least
        $\alpha \cdot n$ polygons.
1  $(V, E, w)$ = buildGraph($G$)
2  $p$ = getUnionOfFacesInGraph($(V, E)$)
3  for $i = 1$ to $\lfloor \alpha \cdot n \rfloor$ do
4      foreach $e \in p$ do
5          if $w(e) == 1$ then
6              $E = E \setminus \{e\}$
7          else
8              $w(e) = w(e) - 1$
9      $V = V \setminus \{v \in V \mid \deg(v) = 0\}$
10     $p$ = getUnionOfFacesInGraph($(V, E)$)
11 return $p$
```

To find the union of the polygons, we apply an algorithm similar to the Graham Scan which finds the smallest polygon containing all points of a set of points [8]. We modified it so that this polygon can only have edges between two corners if there is also an edge between those two corners in $\mathcal{G}$. Starting from a vertex s, instead of finding the vertex t

with lowest angle to the x-axis, as the Graham Scan does, we select the neighbour t of s with the rightmost turn with respect to our path up to now. We are done when the start vertex is reached again. The procedure is stated in pseudo-code in Algorithm 3. All angles are measured counter-clockwise from 0 to 2π.

Algorithm 3: getUnionOfFacesInGraph(Graph G)

```
1  $(V, E, w) = G$
2  $s = s' = l$ corner nearest to the origin
3  $t$ = neighbour of $s$ so that $(s,t)$ has the lowest angle with the x-axis.
   $polygon = (s,t)$
4  $s = t$
5  while $s \neq s'$ do
6      $t$ = neighbour of $s$, so that $\measuredangle(l,s,t)$ is minimal
7      $l = s$
8      $s = t$
9      $polygon$.append($t$)
10 return $polygon$
```

We conclude this chapter with some thoughts on the choice of α. It is easy to understand that lower values often return bad consensual polygons because all input polygons are taken into account. On the other side, if α is to high, then similar phenomenons can occur. Consider a group with many similar rectangles and only one small square inside the rectangles. In the sense of the Pile-Algorithm the polygon consensus is the area covered by the similar rectangles. However, if $\alpha \approx 1$, then the consensual polygon would only represent the small square which is certainly wrong. We recommend values for α near 0.5, but we have not evaluated this advice against lower or higher values in more detail.

Even if the natural description of the Pile Algorithm seems straight away, the implementation of the algorithm is rather complicated. Further, one has to face floating point inaccuracy when handling intersections of polygons as Margalit [10] already pointed out. Besides the restriction of the input, the consensual polygon calculated by the Pile Algorithm often contains a lot more corners than any of the polygon in the input group because many tiny edges are created at the corners and at the intersections between the corners. The polygon consensus could be simplified using track simplification tools like the Douglas-Peucker-Algorithm [6], hence needing one more parameter for simplification. Because of that reason, we will not evaluate the Pile-Algorithm with the real data from the *Building Inspector*, but rather point out that this algorithm might be useful when the polygon consensus has to be precise with respect to the covered area rather than the corners.

5 Cluster-Based Algorithms

The following two algorithms to find a polygon consensus are cluster-based instead of using two-dimensional geometry. The two algorithms are called Voting-Algorithm and Mininum Mean Weight Cycle-Algorithm (Mmwc-Algorithm). They are enframed in two pre-processing steps and one post-processing step:

Outlier Removal Remove remote polygons by clustering the centroids and keep only the clusters with most centroids. The idea behind this step is that sometimes users did not know which building to classify. Hence, there can be independent polygons of several buildings in one group. This step requires a decision on how far away a centroid must lie from the others to be disregarded. Figure 5.1 indicates that this decision is not as trivial as it might sound and that it can affect the result gravely. Of course, there may be other and more advanced methods of detecting outlying polygons, but we did not analyze them in this thesis.

Clustering Cluster the corners of all polygons with a given min-eps. We allow clusters of size one because it is our hope that they are ignored by our algorithms to find a polygon consensus one way or the other. Therefore, we set min-pts $= 1$ for clustering the corners.

Voting-Algorithm or MMWC-Algorithm Construct a graph from the clusters which were calculated in the last step and find a cycle with respect to certain criteria.

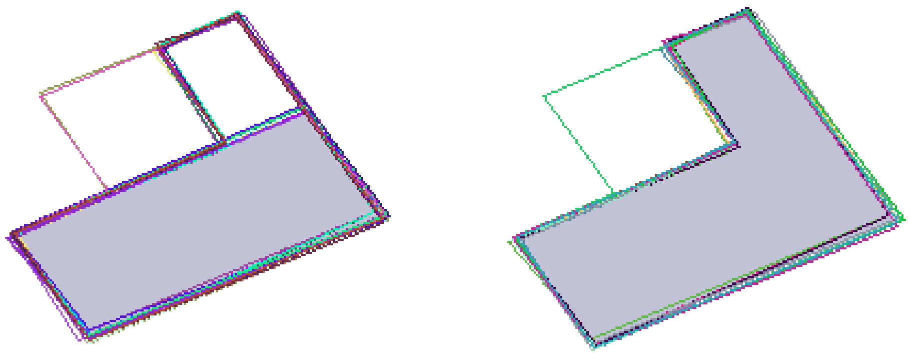
Post-Processing Given the cycle of clusters which is returned by step 3, we now calculate the actual consensual polygon. To do this, we translate each cluster into a corner. Then these corners describe the polygon consensus. The simplest method to translate the clusters to corners is to select the centroid of all corners in a cluster. The disadvantage of using the centroids is that all corners in the cluster must lie equally around that centroid we would consider to be natural. However, one corner which lies far away from the other corners in the same cluster will shift the centroid towards that far away corner. Mostly, such a shift can be avoided by using a better min-eps in the second phase. Instead of using more sophisticated methods of translating clusters to corners, we tried to improve the choice of min-eps in this thesis.

While the actual algorithms do not need any parameters, the preprocessing steps *do* need parameters. If we set min-pts $= 1$ for both preprocessing steps, we still have two different values for min-eps. The head developer of the *Building Inspector* proposed two values which work well with the real data [3]. Because these two parameters are application-specific and sensitive to scaling operations we want the procedure of finding a polygon consensus to be runnable in general without these two parameters. We developed a heuristic which estimates `min-eps` for the Clustering-Phase of the corners, which is described in Section 5.3. In Chapter 6, we will then explore whether the first step is really necessary to produce semantically correct polygons. Figure 5.1 only shows that the results can differ if the outlier removal is skipped, but actually, we are interested in the correctness of the consensual polygon, less whether the polygons are different.

We will learn that the graph G_{M} built by the MMWC-Algorithm is a subgraph of G_{V} which is generated by the Voting-Algorithm. Therefore by showing that there exists always a cycle in G_{V}, we can show that both algorithms always succeed. Given several clusters of a polygon group so that at least one polygon has corners in two of the clusters. Then the Voting-Algorithm constructs the set of edges with the following rules:

1. Select one arbitrary cluster C of corners.
2. Count how often each cluster contains the successor of a corner in C.
3. Select the cluster C' with the maximum count found in step 2. If it turns out that $C' = C$, then take the cluster C' with the maximum count so that $C' \neq C$. If there is no such cluster, continue with step 5.
4. Add the edge (C, C') to set of edges.
5. Repeat these steps for each cluster.

The maximum out-degree in G_{V} is 1 because we select only one successor for each cluster. The following theorem states that there is always a cycle in G_{V} if there exists at least one edge. If the graph G_{V} does not have any edges, than all clusters contain complete and independent polygons. In this case, `min-eps` was obviously wrong and both Algorithms can not return a reasonable consensual polygon.



(a) Allowing the centroids to lie far apart retains the upper polygon.

(b) If only close centroids are allowed, we get another consensual polygon in this case.

Fig. 5.1: The process of filtering outliers can affect the shape of the polygon consensus.

Theorem 5.1. *There is always at least one cycle in $G_V = (V, E)$ if $|E| > 1$.*

Proof. Let G_V be a graph constructed from an arbitrary polygon group. Suppose there is no cycle in G_V. Then G_V is a chain of clusters because each vertex has exactly one successor. Let v be the last vertex in the chain and v' its predecessor. Then there must be polygons containing edges with start point in v' and end point in v. But there can be no polygon with edges starting in v and ending in any other cluster ($\circledast$). Otherwise, v would not be the last vertex in the chain. But when polygons with edges between v' and v exist, then the same polygons must contain an edge starting in v and providing a way back to v'. This contradicts $\circledast$. So G_V must contain a cycle. $\square$

Given Theorem 5.1 we can conclude, that both Algorithms will always be able to find a cycle if the polygon group and the clusters are reasonable. We do not claim that this cycle represents a useful polygon because it is easy to construct instances where the only cycle is of length 2.

5.1 Voting Algorithm

The Voting-Algorithm was first described by Mauricio Arteaga [3]. After applying DBSCAN to the set of corners the graph G_V is generated with respect to the description in the previous section. Lines 1 to 7 in Listing 4 explain the construction of the graph in detail.

After this voting phase, our clusters represent a graph where the set of vertices are the clusters. The out-degree in this graph is at most 1, because we assigned only one successor to each cluster. All we have to do now is to find a cycle in this graph. To find such a cycle we use a trivial corollary from the proof of Theorem 5.1:

Corollary 5.1. *Starting at an arbitrary vertex in G_V with out-degree greater than 0 and traversing the edge progression, we will always reach a cycle.*

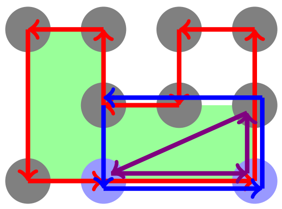
To achieve this, we pick the cluster which contains most polygon corners and traverse its successors. We save all clusters that we see during our progression. As soon as we reach a cluster which we have seen before, we found a cycle. The detailed procedure is given in Pseudocode 4 in lines 8 to 15. Our version of the algorithm is slightly different from the original version by Arteaga. His algorithm starts with an arbitrary cluster. For this to work, all outlying polygons have to be filtered, otherwise the result could be an arbitrary remote polygon. Our version starts with that cluster, that contains the most corners from different polygons. We claim, that this modification renders the initial filtering unnecessary and support this claim by results provided in Chapter 6.

The returned cycle consists of clusters of whose we now have to calculate the actual corners of the consensual polygon. The easiest way to do this is to use the physical centroids of the clusters. In line 9 we return a cycle of length 0 when the most popular cluster does not lead into a cycle. Because we defined polygons to have more than two corners we can interpret such results as indication that the algorithm was not able to find a polygon consensus.

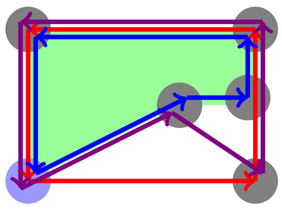
Algorithm 4: The Voting-Algorithm

```
Data: A polygon group $G \in P^+$ and a min-eps for clustering.
Output: A polygon $p$ that is the consensus of $G$.
1  $Clusters$ = DBSCAN(points in $G$, min-eps, 1)
2  foreach $cluster \in Clusters$ do
3      resetVotesOfEachCluster($Clusters$)  // Sets $c$.vote $= 0$ for each cluster.
4      foreach $p \in cluster$ do
5          $target$ = findClusterOfSuccessor($p$)
6          $target$.votes += 1
7      $cluster$.next = cluster with most votes in $Clusters \setminus cluster$
8  $cluster$ = cluster with most corners of distinct polygons
9  if $cluster$.next = Null then return $cluster$
10 $seenClusters$ = [$cluster$]
11 while $cluster$.next $\notin seenClusters$ do
12     $cluster$ = $cluster$.next
13     $seenClusters$.append($cluster$)
14 $cycle$ = $seenClusters$ from first occurrence of $cluster$ to end
15 return $cycle$
```

Unfortunately, the Voting-Algorithm does not always calculate a good consensual polygon. Looking at the situation in Figure 5.2a, where the red polygon was entered 40 times, the violet polygon was entered 25 times and the blue one was entered 35 times, we find that the Voting Algorithm chooses the green shaded polygon. The problem with the green shaded polygon is that it was not present in G and is therefore no polygon consensus. Another problem occurs in Figure 5.2b, where the distribution of votes is the same as in the left figure. Here, the Voting-Algorithm decides for one polygon which has only 35 votes while there is an obvious polygon consensus, namely the red polygon.

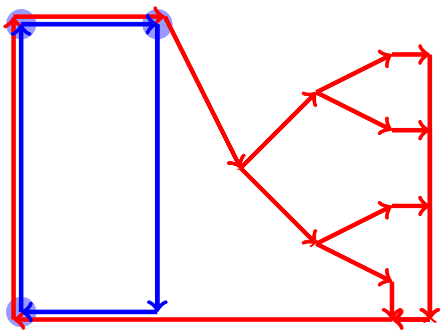


(a) Here there algorithm will return a polygon consensus which was not part of the input. The red polygon has 40 votes, the blue polygon has 35 and the violet polygon has 25 of the votes.

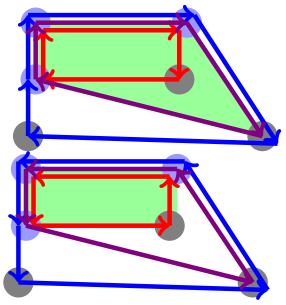


(b) The distribution of the votes is the same is in Figure 5.2a. First, it follows the blue and violet polygons, but then, it decides only for the blue polygon.

Fig. 5.2: Practical relevant issues with the Voting Algorithm. The possible start clusters are marked blue. They contain the highest amount of corners of distinct polygons.



(a) If there is are binary tree somewhere in the graph, the polygon consensus might be backed up by only one of the input polygons.



(b) There are 4 red polygons, 3 violet polygons and 2 green polygons. We can see, that the orientation of the polygons does matter.

Fig. 5.3: Some more examples where the Voting Algorithm may not give the desired polygon consensus. Possible start clusters are shaded blue. Not all clusters are drawn.

This issue can be elaborated to a more theoretical example in Figure 5.3a. In this case there *is* one clear polygon consensus which is the blue one. But there are more red polygons and each of the red polygons is part of a binary tree. The root of this binary tree consists of all red polygons while each leaf is only represented by one red polygon. Then the Voting-Algorithm chooses one red polygon instead of the blue polygon. However, this case is not likely to occur within the *Building Inspector*.

We assume, that all polygons in the input group are oriented in the same direction. In some cases, it does matter whether all polygons are oriented clockwise or counterclockwise. An example is shown in Figure 5.3b. The start clusters stay the same, but the distribution of the votes has changed. The problem concerns all groups where several minor polygons branch from a popular polygon to different clusters, but return to the branch together. In one direction, the main branch has more votes because the other polygons are split. In the reverse direction, their votes add up and can outvote the most popular polygon so their path is pursued. This behavior equals Figure 5.3a, where the popular branch is also abandoned in favor of the minor polygons.

In the next Chapter we will introduce the MMWC-Algorithm which is not orientation-sensitive. It finds the cycle by regarding the graph in a global manner rather than the local aggregation of votes which the Voting-Algorithm does. But exactly this greedy progression of clusters renders the Voting-Algorithm very easy to implement. Despite the different approaches, the MMWC-Algorithm shares some problems with the Voting-Algorithm. We will see in Chapter 6 whether the MMWC-Algorithm can outperform the Voting-Algorithm on real data.

5.2 Minimum Mean Weight Cycle Algorithm

The Voting Algorithm works locally, meaning that the choice of the successor of a cluster c depends only on the situation at c itself. The philosophy of the Minimum Mean Weight Cycle-Algorithm is to work global. Like the Voting-Algorithm, the MMWC-Algorithm takes a list of clusters as input. Then a graph $G_{\mathrm{M}} = (V, E)$ is built, where each cluster represents one vertex. There is a directed edge between two clusters c_1 and c_2, if there is at least one polygon with two successive corners v_i and v_{i+1}, so that v_i lies in c_1 and v_{i+1} lies in c_2. The edge (c_1, c_2) is weighted with the following function w:

$$w((c_1, c_2)) := -\log\left(\frac{\text{Number of polygon edges from } c_1 \text{ to } c_2}{|G|}\right)$$

The more popular an edge is, the lower is its weight given by w. Therefore it is reasonable to find a polygon consensus containing edges with low weight. But the minimum weight cycle would not be a good idea, because a long progression of popular edges could still have a higher total weight than a short progression of unpopular edges. Therefore, we want to find the minimum *mean* weight cycle, that is the cycle in G_{M} that has the lowest average of edge weights. We use the the logarithmic function rather than the a pure linear dependence to degrade unpopular edges even more while preferring the popular ones.

This approach also allows to have several remote polygons in the polygon group. Then the graph consists of several connected components. After finding a minimum mean weight cycle in all of them, we can select the cycle, which has the minimal value among them. In the weight function w we divide through the amount of polygons in the group. This causes connected components with a small amount of polygons to have high weights on all their edges – and therefore a high average weight.

In 1978, Richard Karp published an algorithm to find the minimum mean cycle in a directed, strongly connected graph [9]. For a complete proof of correctness, we refer to his article. The algorithm chooses an arbitrary vertex s in the connected component with n vertices. Then for each vertex v in the connected component, we introduce the variables $F_{v,0}, F_{v,1}, \ldots, F_{v,n}$ where $F_{v,i}$ denotes the minimum weight of an edge progression of length i from vertex s to v. For all v, the initial equality $F_{v,0} = \infty$ holds, except for s with $F_{s,0} = 0$, because only s can be reached from s by using zero edges. All other values $F_{v,i}$ can be computed by using dynamic programming. While computing those values, the predecessors are saved to be able to reconstruct a path of length i from s to a vertex v with the minimal possible weight $F_{v,i}$ later. Given the values $F_{v,i}$ the value of the minimum mean weight cycle is given by the equation

$$\lambda = \min_{v \in \mathcal{V}} \max_{0 \leq k \leq n-1} \left(\frac{F_{v,n} - F_{v,k}}{n - k}\right)$$

The set $\mathcal{V}$ contains the vertices in the connected component. While iterating over all vertices, we do not only save the minimal λ but also the vertex v which causes this value and its maximizing value k. Now we start at v and traverse the progression of length n back to s by using the edges in reverse. Then there must be a cycle of length $n - k$ in this progression, which we find and extract. This cycle is our minimum mean weight cycle. Because the vertices are actually clusters, we can now calculate the centroids of the vertices and return the polygon consensus.

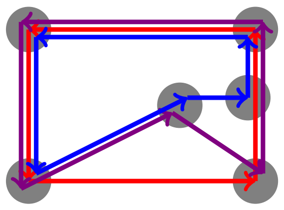
From Theorem 5.1 we know that the graph contains a least one cycle. But for the MMWC-Algorithm to work, we must show that the graph is also strongly-connected.

Theorem 5.2. *Let G_M be the graph generated by the* MMWC-*Algorithm. Then every connected component in G_M is always strongly-connected.*

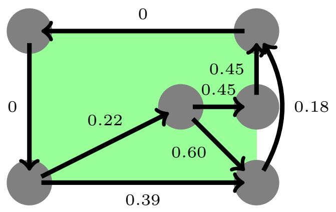
Proof. Suppose there exists a connected component in G_{M} which is not strongly connected. Then there must be a pair of vertices u, v with $(u, v) \in E$, so that u is not reachable from v. If there would be no such pair, then each edge would be part of directed circle, making the connected component strongly connected. Looking at the instruction to build the graph, we know that there exists a polygon so that one edge of this polygon is connecting the clusters respective vertices u and v, else $(u, v) \notin E$. But each polygon contributes a cycle to the graph because the clusters of each polygons form a cycle on it self. So there must be an edge progression from v back to u. This contradicts our assumption. Therefore, every connected component in G is strongly-connected. □

Figure 5.4 illustrates that the MMWC-Algorithm does perform intuitively better than the Voting-Algorithm. As shown in the last section, the Voting Algorithm chooses the blue polygon to be the consensual polygon in the situation depicted in Figure 5.4a, even if the red polygon has more votes. The MMWC-Algorithm, on the other side, generates a graph with the weights depicted in Figure 5.4b. There are three cycles. The first one, representing the result of the Voting Algorithm, has a mean weight of $(0.22 + 2 \cdot 0.45)/5 = 0.224$. The second circle, which was introduced by the violet polygon, has a mean weight of $(0.22 + 0.60 + 0.18)/5 = 0.2$. This leaves the last circle with a mean weight of $(0.39 + 0.18)/4 = 0.1425$, which is the minimum mean weight. However, if the red polygon consisted of k more corners between the two lower clusters, then there would be more edges of weight 0.39. The mean weight of this cycle would consequently approximate 0.39 for a high k. The other cycles would not change and another consensual polygon will be calculated on the same distribution of votes. Although the global approach of the MMWC-Algorithm seems better than the local approach of the Voting-Algorithm, we can construct instances were the result of the MMWC-Algorithm is as undesired as the result of the Voting-Algorithm. Later we will see, whether the global principle gives better results in practice.

Another difference between the MMWC-Algorithm and the Voting-Algorithm is its independence from the orientation of the polygons. While the Voting Algorithm can return different consensual polygons if the orientation of all polygons is changed, the MMWC-Algorithm will always produce the same consensual polygon. This can be proved with the following Theorem 5.3.



(a) We have 8 red polygons, 7 blue polygons and 5 violet polygons. Given this distribution, the graph on the right hand side is constructed.



(b) Now there are three possible circles, the circle with the minimum mean weight is indicated by the green shape. This polygon was in fact entered by most of the users.

Fig. 5.4: In some cases, the results of the MMWC-Algorithm are more satisfiable than the results of the Voting-Algorithm. This is the same example as in Figure 5.2. The decimals are truncated after the second digit.

Theorem 5.3. *Given a clockwise-oriented polygon group $G \in P^+$ and a polygon group G' which contains all polygons in G, but counter-clockwise. Let R be the undirected (multi-)graph using G as input and L the undirected (multi)-graph using G' as input. Then R and L are the same, including the weight of the edges.*

Proof. It is easy to see that R and L contain the same edges without regarding the weight because changing the orientation will not remove or create new edges. So we must only show that the weights of those edges do not depend on the orientation. Thus we have to prove that $w((c_1, c_2)) = w((c_2, c_1))$ for any two clusters c_1 and c_2. Let x be the amount of polygons with a corner p_1 in c_1 and a successive corner p_2 in c_2. By changing the orientation of the polygons, p_2 becomes a predecessor of p_1. Obviously, x did not change, but now it counts the amount of polygons with a corner in c_2 and successive corners in c_1. So $w((c_1, c_2)) = w((c_2, c_1))$ and $L = R$. $\square$

Given this theorem, it is easy to see that the orientation of the polygons in the input group does not affect the actual minimum mean weight cycle because all edges are simply orientated in the other direction. There are no new cycles created or removed by this operation. Unfortunately, this is not a real advantage over the Voting-Algorithm because the data from *Building Inspector* turned out to not include such special groups where the orientation does matter. Although the MMWC-Algorithm does have these theoretical improvements, its implementation is not as trivial as the implementation of the Voting-Algorithm. In the next chapter, we will explore whether this additional effort does pay off against the Voting Algorithm.

A variant of the MMWC-Algorithm uses a probabilistic definition of the weight function, which is more local than the original weight function:

$$w'((c_1, c_2)) := -\log\left(\frac{\text{Number of polygon edges from } c_1 \text{ to } c_2}{\text{Voting corners in } c_1}\right)$$

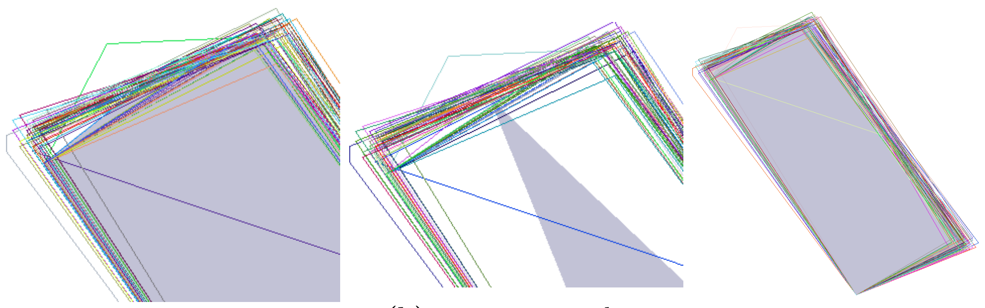
We did not use this definition, because it does not work well with several connected components. If there exists a connected component which consists of only one polygon, then its own cycle has weight 0. In case there is another connected component like the one of Figure 5.4 with an obvious polygon consensus, the cycle of weight 0 would be returned, backed up by only one polygon. Another disadvantage of w' is that the orientation of the polygons does matter.

5.3 Choice of Parameters

In the last two sections, we assumed that we already have a suitable min-eps to cluster the vertices. If the user of the algorithms knows the geographical background of the polygons, it may be easy to find an min-eps which creates reasonable clusters. It is also simple to interpret the magnitude of min-eps to be some sort of geographical information about the data. But in some cases, it might be useful to estimate min-eps, for example if one does not have a geographical context or the scaling of the polygons has changed. In this section, we propose an idea to find min-eps automatically.

The choice of min-eps affects the resulting consensual polygon. If the value is chosen too low, then there are too many clusters and the polygon consensus may represent exactly one of the input polygons (Figure 5.5a). On the other hand, higher values of min-eps can make two different clusters merge into one cluster as Figure 5.5b shows, where the single corner in the center of the line segment causes the left and right clusters to collapse. Last but not least, Figure 5.5c shows a good choice of min-eps.

In this section we assume that if a polygon group $G \in P^+$ contains an intuitive polygon consensus, then this consensus does only use clusters with more than $|G|/2$ corners. Otherwise, we argue that there is no obvious polygon consensus in the group. Therefore, we allow only clusters of size $|G|/2$ or greater from now on. When we consider the amount of clusters in a polygon group G as a function of min-eps, a diagram like to one in Figure 5.6 results. We call the input parameter of that function ϵ to distinguish it from the min-eps used for clustering. As long as min-eps is too small, there are no



(a) min-eps is too small. **(b)** min-eps is too big. **(c)** min-eps is is suitable.

Fig. 5.5: The same input set is shown with different values of min-eps. In the left picture, min-eps is too high, in the center picture, min-eps is too low. The right picture shows the result with accords with the expected result for this polygon group.

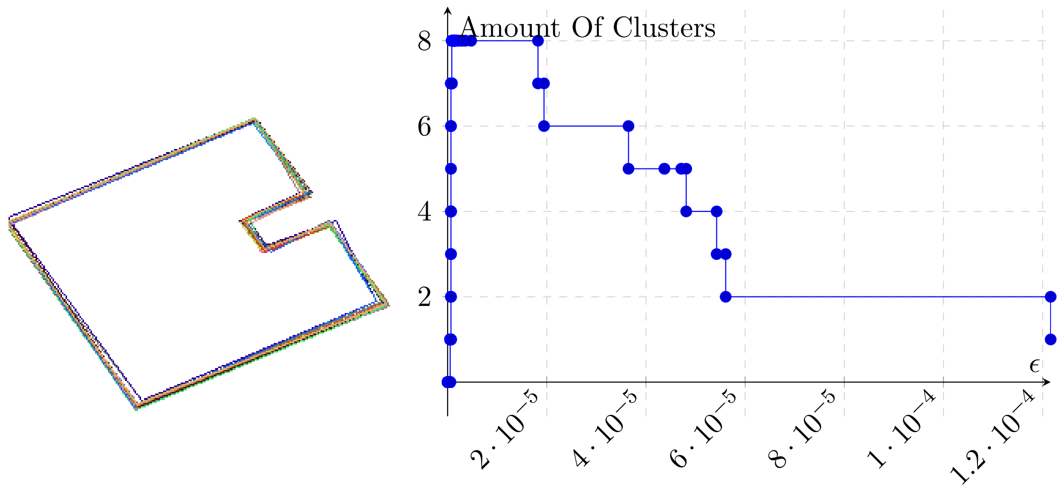


Fig. 5.6: Given the polygon group on the left side, the diagram on the right side results. We can see, that the desired amount of eight clusters is reached fast and it is stable. The next stable plateau is reached when the clusters on the inside merge together. Because the footprint is nearly square, there is a long plateau with an amount of two clusters before collapsing into one cluster.

suffiently congregations to form a cluster of size $|G|/2$. If min-eps is increased, the amount of clusters increases too, until a threshold is reached. Then, while increasing min-eps further, several clusters will be merged and the amount of clusters decreases. Of course, it is possible that the amount of clusters ascends again, but finally, min-eps is large enough so that all points in G fit into one cluster.

We also assume that most polygons in G will have the same amount of corners – namely the amount of corners the polygon consensus has. So from the diagram in Figure 5.6 only those plateaus are interesting where the amount of clusters is the median of the corners in G. But there may be several of them. By selecting a min-eps in the longest plateau of them, we find a value which represents the right amount of corners and the most stable of these plateaus.

The naive way to implement this algorithm would be to start with a low value of min-eps, then apply Dbscan, count the clusters and increase min-eps. But this would lead to the question of the magnitude Δ by which we increase min-eps in each iteration. The value of Δ again depends on contextual data like the scaling factor and the distribution of the corners. Too small values of Δ lead to longer running times and values to high may oversee some plateaus. We present a better way to construct the Figure 5.6 without these difficulties.

A change of the amount in Figure 5.6 at the value ϵ implies that ϵ represents exactly a distance between to corners in the input list. The inversion of this implication is not true because there may be distances between pairs of points which do not change the amount of clusters. But this implication is enough to reduce the calls of Dbscan. We calculate the distance between all pairs of corners and sort them. If there are n corners, than we have $\mathcal{O}(n^2)$ distances. Now we can take those distances and give them as values for min-eps to DBSCAN. If we count the clusters after each invocation of DBSCAN, we also obtain the diagram above. This makes roughly $\mathcal{O}(n^2)$ calls to DBSCAN and a running time of $\mathcal{O}(n^4)$ in total if we implement DBSCAN naively. A better running time to generate the diagram above can be used if we build the clusters successively. Let C_i be the set of clusters with using min-eps $= d_i$ where d_i is the i-th distance in the sorted list of distances. In the next step, the distance d_{i+1} is used. Three events can now occur: Two clusters merge because d_{i+1} links their connected components. Or a new cluster is created because the min-pts–threshold is reached for an area. Or simply nothing happens. We see that we do not need to call DBSCAN with d_{i+1} as min-eps. Instead we can evaluate which event occurs after each step. To achieve this, semi-dynamic sets become handy.

A *semi-dynamic set* is a data structure which manages sets, often it is also called *disjoint-set data structure* [5]. Apart from the default constructor, it provides four operations which are closely related to graph operations.

insertSet(e) Takes an element e and inserts the set $\{e\}$ into the data structure. In graph theory, a new vertex e is added to the graph with $\deg(e) = 0$.

findSet(e) Returns the set in which the element e currently resides. Speaking of graphs, this method returns the connected component in which e lives.

unionSets(s_1, s_2) Replaces the sets s_1 and s_2 in the data structure by their union $s_1 \cup s_2$. Or, in graph terms, adds an edge between the connected components s_1 and s_2.

size() Returns the current number of sets in the data structure or the number of connected components respectively.

To start, we insert all n corners into our semi-dynamic set D. If we now called D.size(), the answer would be n, which is the same as calling DBSCAN with min-eps $= 0$ and min-pts$= 1$. As we have set min-pts to $|G|/2$, we modify the size()-operation accordingly. Now we can use our semi-dynamic set to execute DBSCAN step-by-step with increasing min-eps-values. Everytime when a plateau with the desired amount of corners ends, we check whether this plateau is the longest we have seen so far. If this is the case, the current sets in the semi dynamic set are stored to be used with the Voting-Algorithm or the MMWC-Algorithm later. The actual value of ϵ in the moment of saving the clusters does not matter with regard to our interpretation of the plateaus. So we take the lowest value possible. Keep in mind, that by using other, equally reasonable values for ϵ, the amount of corners in the clusters can change and therefore, the centroids can change. The actual min-eps value is no longer needed, expect for debugging purposes. Naive implementations of the semi-dynamic-set–operations have a running time of $\mathcal{O}(n)$ where n is the amount of corners. This running time suffices for our application so we did not improve it in order to keep the algorithm simple. The overall running time of the min-eps-estimation is now $\mathcal{O}(n^3)$ which is faster than using DBSCAN all over. A detailed description of the algorithm is given in Pseudocode 5.

Of course, this solution is not robust in difficult polygon groups. In the next chapter, we will also evaluate how this procedure to find a min-eps automatically performed on the real data of the *Building Inspector* and explain those examples, in which this method did not work.

Algorithm 5: An algorithm to estimate min-eps.

```
Input: A group of polygons $G$
Output: Clusters obtained by estimating an min-pts-value
1  $D$ = new Semi-dynamic Set
2  $x_{\text{med}}$ = median of corners
3  foreach corner $c \in G$ do
4      $D$.insertSet($c$)
5  Let $Distances$ be a sorted list of all distances between pairs $p_1$ and $p_2$ of corners.
6  $longestDist = 0$
7  $plateauStarted$ = false
8  $plateauStart = 0$
9  $clusters = \emptyset$
10 foreach distance between $p_1$ and $p_2 \in Distances$ do
11     $s_1 = D$.findSet($p_1$)
12     $s_2 = D$.findSet($p_2$)
13     $D$.unionSets($s_1$, $s_2$)
14     $count = D$.size()
15     if $count = x_{\text{med}}$ and $plateauStarted$ = false then
16         $plateauStarted$ = true
17         $plateauStart$ = distance between $p_1$ and $p_2$
18         $clusters$ = Sets in $D$ with size greater or equal than $x_{\text{med}}$.
19     if $count \neq x_{\text{med}}$ and $plateauStarted$ = true then
20         $plateauStarted$ = false
21         if distance between $p_1$ and $p_2 - plateauStart > longestDist$ then
22             $longestDist$ = distance between $p_1$ and $p_2 - plateauStart$
23 return $clusters$
```

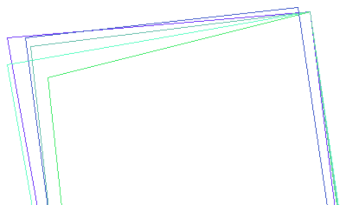
6 Practical Results

We have implemented all algorithms presented in the last chapter. In this chapter, we take a look on the results and the performance of these algorithms in connection with the *Building Inspector*.

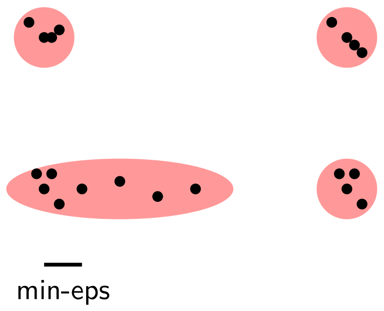
6.1 The Input Data

We tested the data with real polygon groups from the *Building Inspector*. As described in Chapter 1 the users were asked to classify the image-processing-detected polygons to be true, need a fix or to be completely false. We received 5,834 polygons which were classified to need a fix. For each of those 5,834 polygons, we had up to 70 user fixes per polygon. Therefore, our base data consisted of roughly 60,000 user fixes. The actual image-detected polygons were only used as a reference in the evaluation later.

As a consequence from the interface described in Chapter 1, there are two structural observations on the polygon fixes. First, attention must be paid to such cases depicted in Figure 6.1a. Obviously, the image-detecting algorithm figured out one corner correctly. This corner was not moved by most of the users and exists therefore in nearly every user-contributed polygon. To handle such cases correctly, we must not put the raw corners in sets. If we did, the sizes of the clusters were not correct. So we gave individual identification numbers to the corners because sets only allow unique elements.



(a) The image-processing algorithm obviously detected one correct corner. This corner was then never moved by most of the users.



(b) In this group, there are four clusters, but one user did not delete superfluous corners. This influenced the shape and consequently the centroid of the lower left cluster.

Fig. 6.1: This figure shows two of the structural properties we found in the real data.

Processor	Intel Core i7
Processor Frequency	4 GHz
Processor Cache	8 MB
RAM	16 GB
Operating System	Linux Mint 17.2 64 bit
Linux Kernel Version	3.16.0-38
Hard Drive	SSD
Python Version	3.4.3

Tab. 6.1: Description of the computer used to run the algorithms.

A second problem occurs when the a user simply pushes the corners on the delimiting line of the building rather than deleting superfluous corners. Of course such cases will not affect the consensual polygon in general because most users deleted such corners. However in some cases, this behaviour caused some clusters to be too big which is illustrated in Figure 6.1b. As long as there are much more corners at the right place, the centroid of that cluster also stays in the right position. But in some cases, there were not enough correct polygons to compensate such problems. There are two solutions for this problem. The easiest, yet unsatisfying, solution is to change the value of `min-eps` – which likely leads to problems in other places. Or a more advanced method of translating the clusters to corners could be used to solve the issue. However, we have not evaluated these solutions in this thesis.

6.2 Implementation

For the implementation of the algorithms above we used the Python programming language. The algorithms run on a desktop PC, whose technical details are described in Table 6.1. We did not use libraries to represent the graphs and to run algorithms on them. Solely the parsing of the input file in JSON-format and the generation of the images you see in this thesis were left to the built-in python modules.

The running time of the algorithms was measured with the linux command line tool `time`. The running time to evaluate 8,534 polygon groups was always below three minutes. In this three minutes, a file containing all 8,534 groups was parsed, the consensual polygon was calculated and the result was written to a text file, one text file per group. Additionally, debugging information like the used parameters and amount of polygons was written to the standard output.

6.3 Process of Evaluation

After running each of the algorithms (except the Pile Algorithm) on every polygon group, we evaluated the results. This evaluation consisted of two phases. In the first phase, we decided for each existing polygon whether it is semantically true, meaning that the polygon does in fact represent a building. In order to do this, we implemented a tool which projected the polygons on the historic maps. A polygon is semantically correct if it has the same amount of corners as the building it surrounds. Further, the inscribed shape must have a background color other than white and this shape must not be disconnected by a wall. Some examples are illustrated in Figure 6.2. This phase was rather time-consuming, we therefore selected randomly 200 polygon groups so that each group contained between seven and sixteen polygons. The 200 groups were parted into two bundles of 100 groups which were checked separately. During this manual evaluation, we did not know how the polygons were generated. There are eight sources a polygon can come from:

detected The polygon was generated by the Image-Processing-Tool.

user The polygon was entered by a user.

voting-clean Result of the Voting Algorithm with removing the outlying polygons.

voting-raw The Voting-Algorithm without the first preprocessing step.

voting-autoeps The Voting-Algorithm, min-eps was auto-estimated, no preprocessing.

mmwc-clean Result of the MMWC-Algorithm with removing the outlying polygons.

mmwc-raw The Voting-Algorithm without the first preprocessing step.

mmwc-autoeps The MMWC-Algorithm, min-eps was auto-estimated, no preprocessing.

For all clustering processes we took the constants provided by Arteaga because he claims that they work best on the data [3].

It is reasonable to conduct this step by humans because if it would be possible to check the polygons automatically against the historic maps, then the classification of the polygons in the first step of the crowd-souring project could have been omitted. It is important to note that we did not evaluate the accurateness of the polygons.

From the first phase, we expected all of the detected polygons to be semantically incorrect – otherwise they would have been rated with "yes" in the first place. The amount of correct user-contributed polygons should be fairly high, but not one hundred percent because we knew that some users did not interpret the maps right or did not delete unnecessary corners. Then we hoped that the algorithms without outlier removal are equally successful as the algorithms with outlier removal because we claimed that this step is not necessary. At last, we wanted to find out how the min-eps-estimation would perform because we had no idea how it will react to the real data. We knew already that the estimation will fail on some groups but we were anxious to see how many good polygon consensus this heuristic would give.

In the second phase, the accuracy of the polygons was rated. This could be done automatically because all edges should lie on walls. These walls are painted black. Consequently, the pixels behind the edges should be very dark. Using the method `Color.getBrightness()` of the .NET-Framework [1], we obtained a value between 0.0 and 1.0 for each pixel, where 0.0 represents a completely black pixel. Given a polygon,

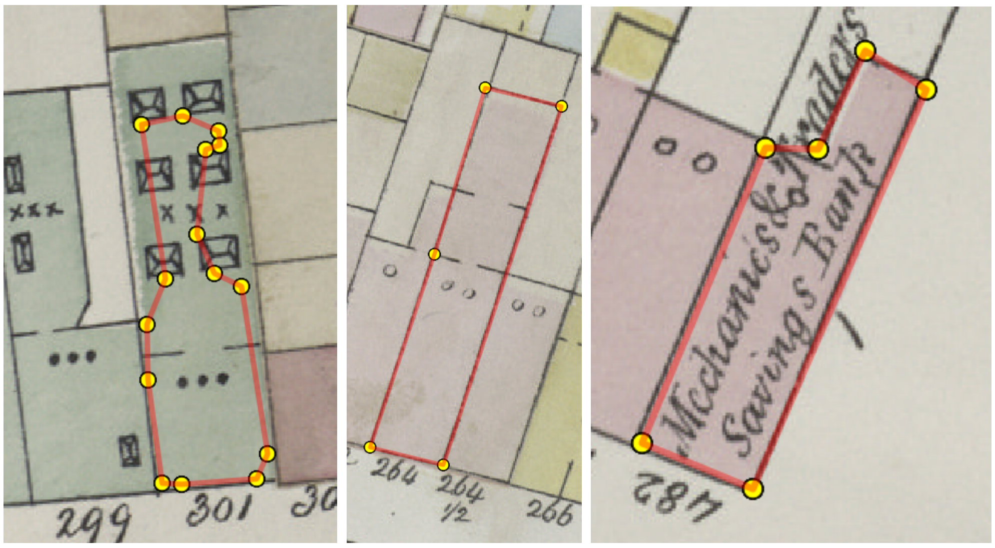


Fig. 6.2: The instance on the left is semantically incorrect because the polygon is certainly not representing a building. The image in the middle is also not correct because it has one corner too many. Even if there are two inaccurate points on the right picture, this polygon is semantically right because it covers the correct shape and has the proper amount of corners.

our tool found the average brightness for each edge. Then the average of all edges of the polygon was calculated. All edges are weighted equally no matter how long the respective edges are. Given this average brightness, we can rate the accuracy of the polygon. Low values mean a high accuracy, high values mean a low accuracy. This result does not depend on the semantic correctness of a polygon as Figure 6.3 shows. The left and the center images show semantically correct polygons, but the accuracy of the center image is very bad compared to the left picture. The right picture shows a accurate polygon, yet, this polygon is semantically wrong.

Our hope with the results of this phase was that the polygons given by our algorithms were more accurate than the user polygons. If that is the case, then using those algorithms is even then useful if the users entered only semantically correct polygons.

6.4 Results of the Evaluation

In the first phase, every polygon was reviewed by three persons. Of the three votes given by the three persons, the majority determines if the polygon is semantically correct. The results are grouped by source in Figure 6.4. It shows that our exceptions turned out to be true. In fact, all detected polygons were wrong, while most of the other polygons are correct. It became also clear that by using one of the algorithms with fixed min-eps-value the probability to obtain a correct polygon is higher than by simply select one of the user polygons to be the consensual polygon.

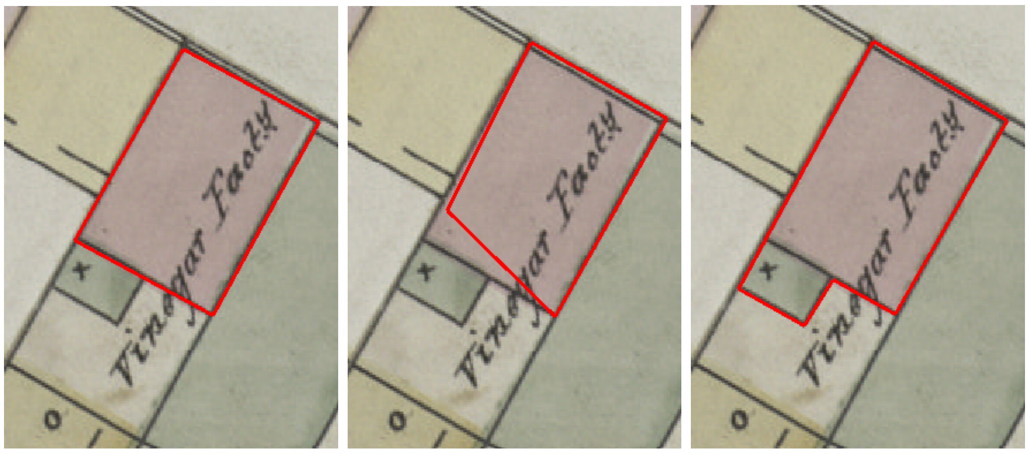


Fig. 6.3: The leftmost polygon is correct and has an average brightness of 0.36, the polygon in the center has an average brightness of 0.48, but it is semantically correct. The last polygon has an average brightness of 0.38, which is pretty small. However, the polygon itself is not useful.

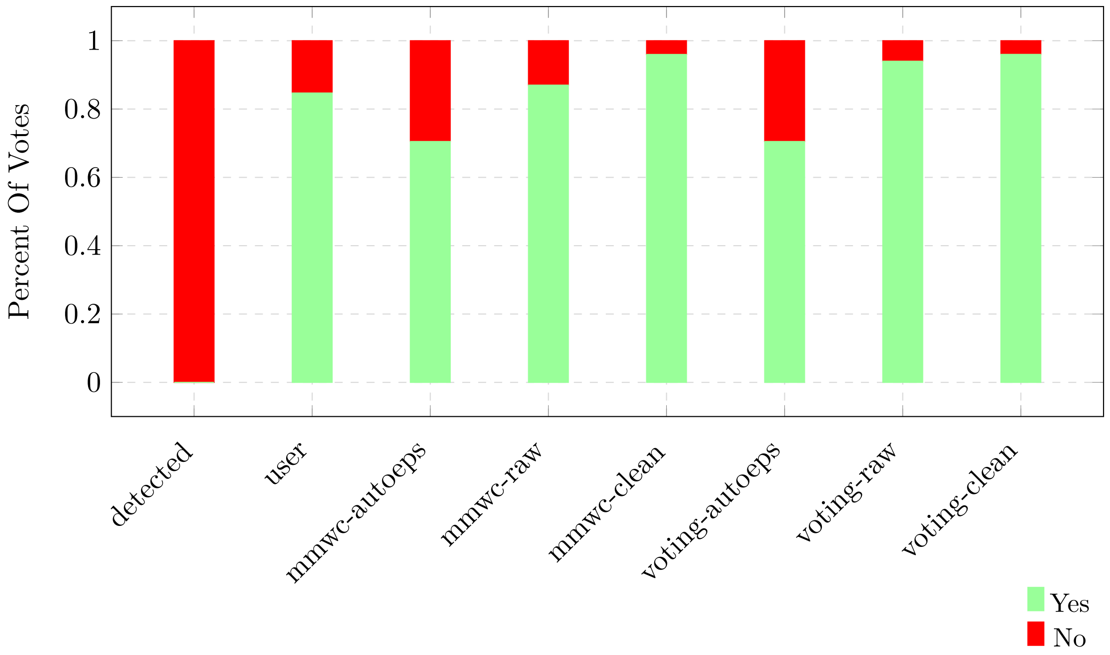


Fig. 6.4: The diagram shows the distribution of the Yes/No votes for each of the sources. All detected polygons were semantically wrong while most of the user polygons were correct. However, the amount of correct polygons could be increased by using one of our algorithms except for the algorithms estimating `min-eps`.

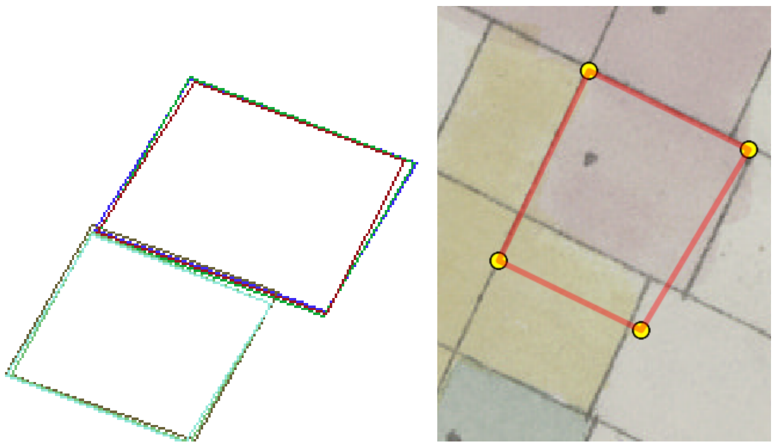


Fig. 6.5: In this case, all polygons have 4 corners. When using a autoeps-variant, the longest plateau with 4 clusters is searched. This state can only be reached when two clusters merge together, resulting in the image on the right. Naturally, there would have been six clusters.

It is also interesting to examine the question whether the algorithms agreed in a particular polygon group. Of the 200 investigated groups, in 122 cases all algorithms produced a semantically correct result. In 45 of the remaining groups, the autoeps-algorithms were not able to produce a semantically correct polygon while the results of the other algorithms were correct. All of those 45 groups were reviewed and we found that in 42 cases, the polygons described different buildings. Such a situation is depicted in Figure 6.5. The other three groups, on which the autoeps-variants failed, contained a large amount of polygons with too many corners that the users simply put on the border. This is the same problem that Figure 6.1b concerns. In those 45 groups, the polygon consensus itself was ambiguous. In such cases, the autoeps-variants gets confused and returns a wrong polygon which can interpreted as an indicator that there is no obvious polygon consensus. The other algorithms, however, return a semantically correct polygon based on the user inputs because they can choose any polygon which is backed up a sufficient amount of users. In such cases there is no indicator that there was actually no real polygon consensus in the group. Yet, there were two very interesting groups in which only the auto-eps variants found a correct polygon. In this case, the min-eps-value of Arteaga was too high causing two clusters to merge while most of the users recognized both near corners as can be seen in Figure 6.6. Therefore, the autoepsvariants using the median worked well.

In 17 cases, the preprocessing turned out to be necessary. The algorithms without removing the outlying polygons did not find a correct polygon. In the remaining 16 groups, there was no polygon consensus because the user fixes were parted in several equally often selected polygons. Because of that, the choice which consensual polygon to take was arbitrary and so it was different for each algorithm.

We conclude from these observations that if we had removed the outlying polygons in the autoeps-variants, they would have worked better. We also conclude that the

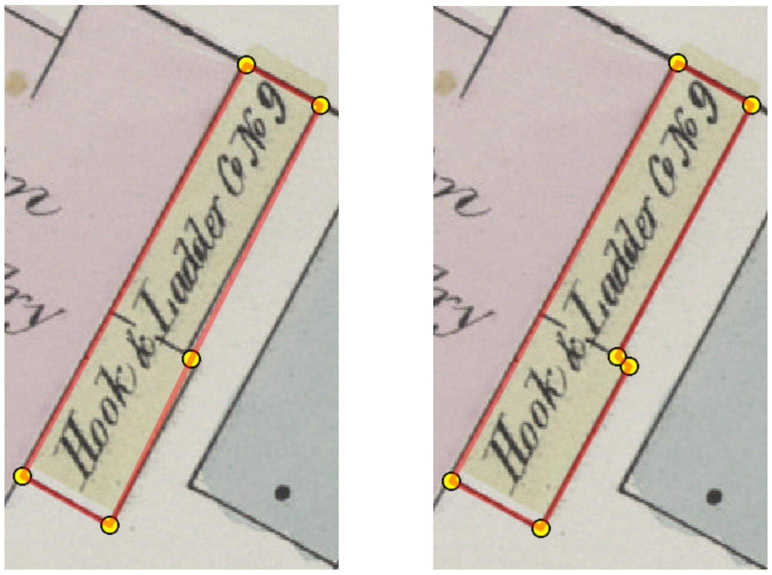
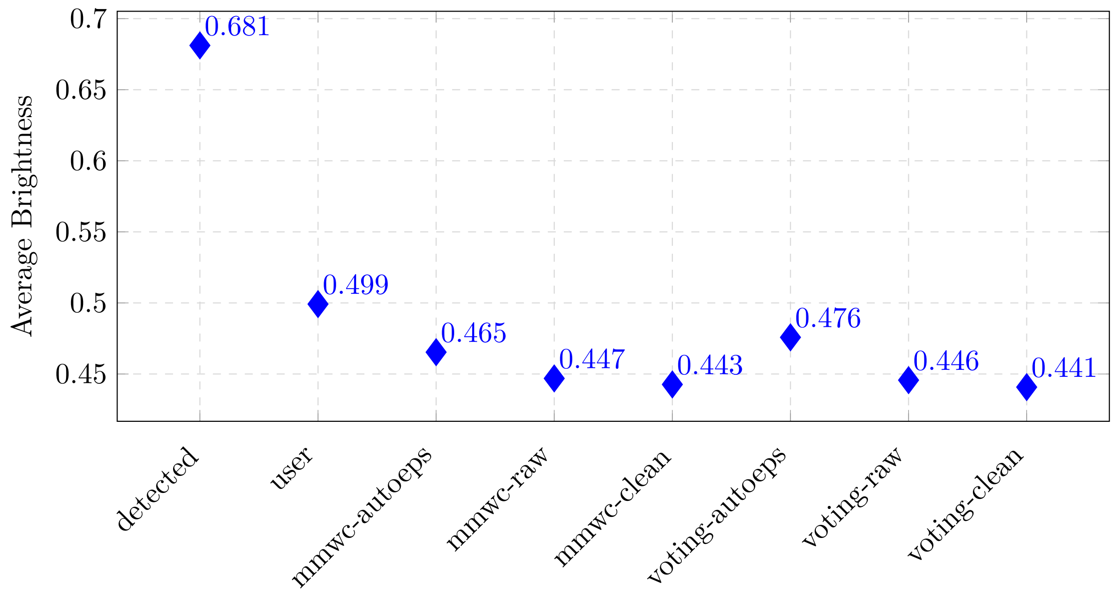


Fig. 6.6: This is an example for the disadvantage of a constant min-eps for all groups. On the left side, the result of the Voting-Algorithm with Preprocessing and Clustering is printed. The right images shows the result of the auto-eps variant of the Mmwc-Algorithm.

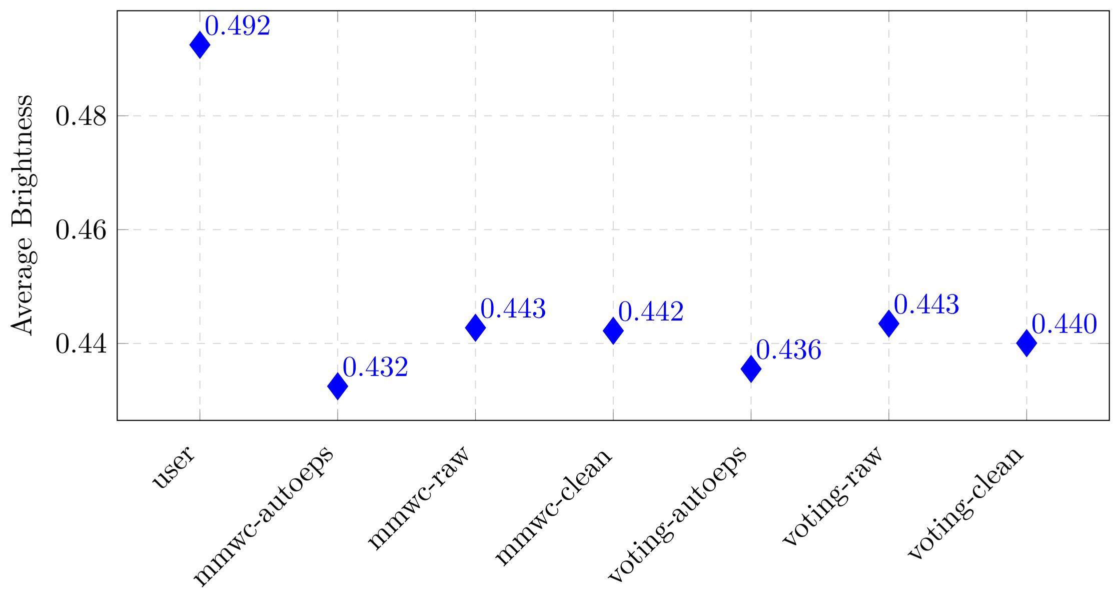
fixed min-eps for clustering the corners delivered more correct polygons than the user fixes. Consequently, one preparing step is needed, either outlier removal or clustering the corners. Based on the dataset, one of them might be easier to implement. With the data from the *Building Inspector* it is obviously easier to cluster the corners with a fixed min-eps than to remove the outliers. But on other datasets it may be easier to remove the outliers than to give a fixed min-eps. This choice can be made based on the kind of data.

If we plot the average brightnesses of each source, we obtain the diagram of Figure 6.7a. It is interesting that the detected polygons have the worst brightness of all because the image-detecting algorithm works by identifying dark lines the maps. The result of this evaluation implies that if the image-detecting algorithm fails then it fails completely. We see also, that the brightnesses of all other sources are lower and, therefore, the polygons are more accurate. However, the values of these sources are not interpretable straight-away because they also include those polygons which were semantically wrong. We are only interested in the accuracy if the polygon itself is reasonable. Figure 6.7b presents also the average brightnesses, but now only of those polygons which were semantically correct.

Given the results of the brightness, it is still arguable to select the user-polygon which has the lowest brightness and consider that as the polygon consensus. In fact, in 185 of the 200 groups, this strategy would yield a semantically correct polygon. In 121 of those 185 groups, the algorithms found a more accurate, semantically correct polygon then the users. This result strengthens our thesis that using the algorithms *is* better then selecting the darkest user polygon.



(a) The average brightness of the resulting polygons for each source. These values include all polygons, even if they were semantically wrong.



(b) The average brightness of the resulting polygons for each source, using only the polygons that are semantically correct. This image does not show the detected polygons because there were no correct polygon consensuses from the image-processing algorithm.

Fig. 6.7: The average brightnesses using all polygons versus only the correct ones.

To conclude the evaluation, we finally want to know which algorithm is the best to use. We already know that the choice whether to remove the outliers or to cluster the corners using a fixed min-eps depends on the kind of data. Therefore, we simply count how often the Voting-Algorithm or the Mmwc-Algorithm found more accurate polygon consensus without regarding the specific variant. It turns out, that on our data, the Voting-Algorithm produced in 93 cases the most accurate and correct polygon and the Mmwc-Algorithm produced in 83 cases the most accurate polygon. In the remaining cases, they found equally accurate polygon consensuses. This implies the both algorithms are equally suitable to work on the given data. Despite the theoretical disadvantages of the Voting-Algorithm we recommend using it because it is easier to implement and less complex than the Mmwc-Algorithm.

7 Conclusion

Among the tasks to be done for digitizing history maps of New York City, the extraction of building-footprints can be found. Users were asked to correct the polygons which were detected by a image-processing-algorithm. This thesis proposed several algorithms to obtain one polygon consensus from a group of user-contributed polygons. Some of these algorithms can be used indeed to generate a reasonable polygon consensus representing building outlines.

The Pile-Algorithm uses a geometric approach. It interprets the polygons as independent geometric structures and finds the area that is covered by $\alpha \cdot n$ edges, where α is a parameter given by the user and n the amount of polygons. The big disadvantage of the Pile-Algorithm is its behaviour at the corners of the polygon consensus. Because of the geometric approach, many additional corners are produced in this area. Therefore, the Pile-Algorithm is not suitable for calculating polygons representing buildings.

The other two algorithms cluster the polygons first. Consequently, the input polygons are no longer independent geometric structures, but rather related clusters. Then both algorithms build a graph of these clusters and search for a cycle in the graph. After the post-processing, that cycle represents the consensual polygon. We have seen that the Mmwc-Algorithm returns in some theoretical cases better results than the Voting-Algorithm. However, the Voting-Algorithm is easier to implement. Both algorithms need a constant to cluster the polygons. We proposed a heuristic method to find this constant automatically for each polygon group.

We did evaluate the cluster-based algorithms with real world data from the *Building Inspector* and found that one of the pre-processing steps can be skipped without loss of quality. Because these pre-processing steps are very different, it may be easier to apply one or the other to a dataset. Using our results, one can decide, which of those pre-processing step is more suitable for the data at hand. Finally, we concluded that the Mmwc-Algorithm does not have a practical advantage over the Voting-Algorithm. We therefore recommended to use the Voting-Algorithm because of its simplicity.

There are more topics to be explored. The most interesting question is a profound analysis of the pre-processing step to remove remote polygons and how it affects the consensual polygon. The evaluation showed that the results with and without pre-processing can differ. Another issue is the interpretation of the weight function in the Mmwc-Algorithm. We found a weight function that works well, but it would be nicer to be able to interpret the weights with regard to the actual polygon group. Another subject of further work is to find a better min-eps-estimation which does not get as easily confused as our algorithm. Or it can give a warning instead of a wrong polygon if it gets confused. Generally, it would be nice if all our algorithms could rate their results in some way so that the user can filter bad ratings and review those groups manually.

We also suggest to analyze the realization of the crowd-sourced part of the *Building Inspector*. One major problem was, that the users did not know which building to select because the shown polygon covered several footprints. This lead to many outliers we had in the groups. A possible solution is to simply show the centroid of the image-detected polygon and ask the users to select the building containing that centroid. Then all polygons would describe the same footprint. Also, the definition of buildings are not really precise. The definitions say that a building must have a background color. But there are many footprints having two or more background colors. Further, some footprints are completely dotted and have a background color. We do not know whether these footprints are buildings, too. With more precise definitions, the user-fixed polygons would have been much more homogenous and the calculation of the polygon consensus a lot easier.

On June 18, 2016, a paper with the title "Polygon Consensus: Smart Crowdsourcing for Extracting Building Footprints from Historical Maps" was submitted to the 24th ACM SIGSPATIAL 2016. It was written alongside this thesis by Benedikt Budig and Thomas van Djik. I provided the implementations and the data used in the last chapters.

Bibliography

[1] Color.GetBrightness()-Method. https://msdn.microsoft.com/en/us/library/system.drawing.color.getbrightness%28v=vs.110%29.aspx. Accessed on June 9th 2016.

[2] New York Building Inspector. http://buildinginspector.nypl.org/. Accessed on May 23th 2016.

[3] Mauricio Arteaga. Finding shape consensus among multiple geo polygons. http://nbviewer.jupyter.org/gist/mgiraldo/a68b53175ce5892531bc. Accessed on May 23th 2016.

[4] Mauricio Arteaga. Historical map polygon and feature extractor. In *Proceedings of the 1st ACM SIGSPATIAL International Workshop on MapInteraction*, pages 66 – 71, 2013.

[5] Thomas H. Cormen, Charles Eric Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to algorithms*. MIT press Cambridge, 3rd edition, 2009.

[6] David H Douglas and Thomas K Peucker. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *Cartographica: The International Journal for Geographic Information and Geovisualization*, 10(2):112–122, 1973.

[7] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In Usama M. Fayyad Evangelos Simoudis, Jiawei Han, editor, *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD-96)*, pages 226 – 231. AAAI Press, 1996.

[8] R.L. Graham. An efficient algorith for determining the convex hull of a finite planar set. *Information Processing Letters*, 1(4):132 – 133, 1972.

[9] Richard M. Karp. A characterization of the minimum cycle mean in a digraph. *Discrete Mathematics*, 23(3):309 – 311, 1978.

[10] Avraham Margalit and Gary D. Knott. An algorithm for computing the union, intersection or difference of two polygons. *Computers & Graphics*, 13(2):167 – 183, 1989.

Erklärung

Hiermit versichere ich die vorliegende Abschlussarbeit selbstständig verfasst zu haben, keine anderen als die angegebenen Quellen und Hilfsmittel benutzt zu haben, und die Arbeit bisher oder gleichzeitig keiner anderen Prüfungsbehörde unter Erlangung eines akademischen Grades vorgelegt zu haben.

Würzburg, den 20. Juni 2016

....
Fabian Feitsch