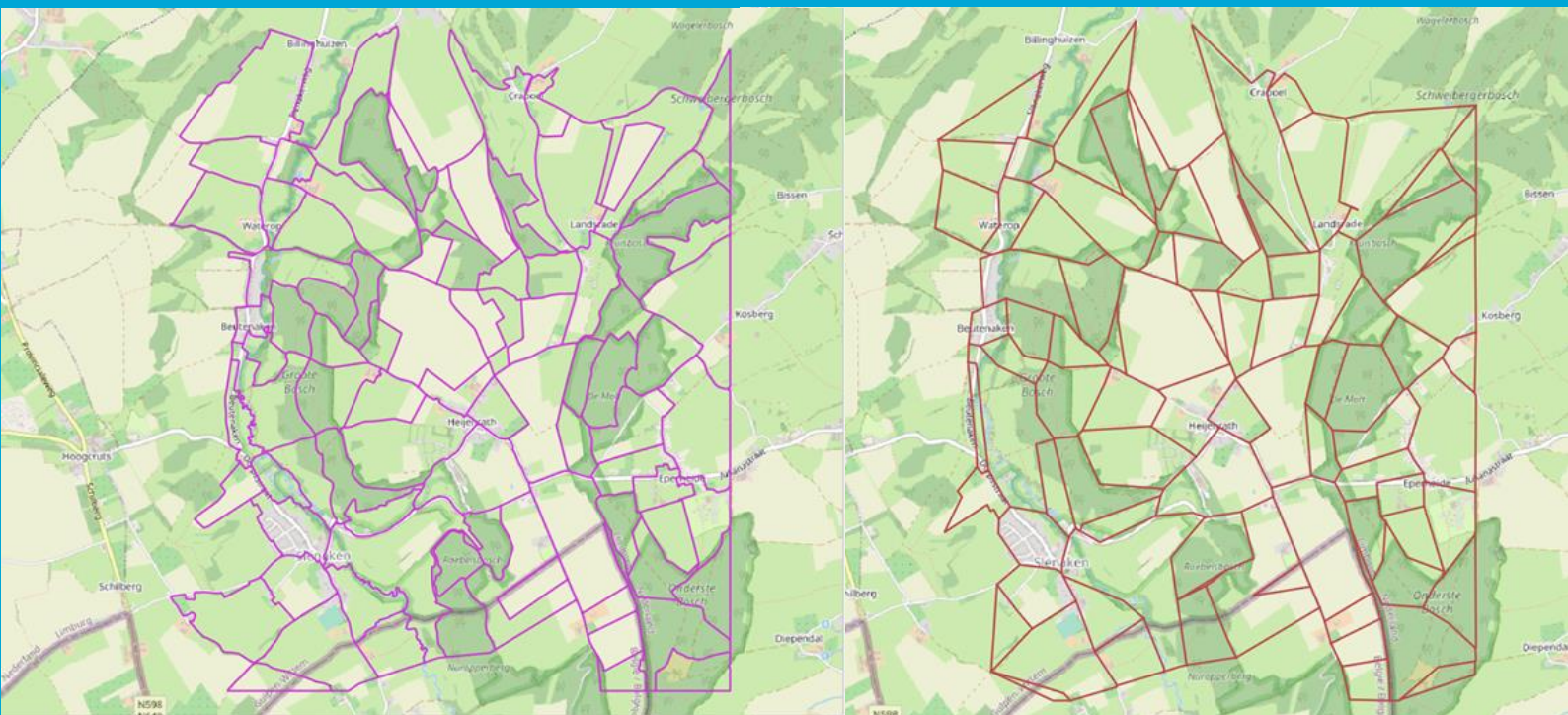


MSc Thesis in Geomatics

Improving the content of Vario-Scale Maps- An analysis into the generalization of border structures

Mihai-Alexandru Erbaşu



2023

MSc thesis in Geomatics

**Improving the content of Vario-Scale
Maps - An analysis into the generalization
of border structures**

Mihai-Alexandru Erbasu

January 2023

A thesis submitted to the Delft University of Technology in
partial fulfillment of the requirements for the degree of Master
of Science in Geomatics

Mihai-Alexandru Erbasu: *Improving the content of Vario-Scale Maps - An analysis into the generalization of border structures* (2023)

© This work is licensed under a Creative Commons Attribution 4.0 International License. To view a copy of this license, visit <http://creativecommons.org/licenses/by/4.0/>.

The work in this thesis was carried out in the:



Geo-Database Management Centre
Delft University of Technology

Supervisors: dr.ir. B.M. Meijers
Prof.dr.ir. P.J.M. van Oosterom
Co-reader: Drs. C.W. Quak

Abstract

As the way we interact with maps keeps changing, so do the maps change alongside. And it can be easily pointed out how these changes come alongside a large number of advantages for the average map user, such as quick access to data or the ability to view more or less of the Earth's surface with just a mouse scroll, as well as for specialists such as cartographers or spatial data analysts, as it is now easier than ever to manipulate complex data. That being said, the challenges have also shifted, from the expertise of the map maker to the software solutions which now do all the work.

One of the many challenges imposed by the aforementioned is represented by the way the map generalization process is achieved. This graduation project serves as a continuation to the countless amount of research which has already been performed in this field, with a focus on the niche world on Vario-Scale Maps, and in particular how borders are handled in this generalization process.

There is already a large number of different solutions available, some of them being considered as standard and used by some of the biggest players in the world of geo-information. However, it seems that no single one solution is a 'silver bullet', as they all have their advantages and disadvantages, as well as cases where one generalization workflow is clearly more suited than others.

Considering the actual status quo of the industry, this thesis will take a look at some of these already available solutions on the market, both individually as well as together, and will try to answer the following research question: *To what extent can multiple line-generalization algorithms be (simultaneously) introduced in the Vario-Scale structure such that they preserve the topology and enable an optimal line density (while trying to preserve the characteristics of the initial shape as well).*

To reach an answer, it is necessary to start first from the lowest level, with understanding how line generalization function in different situations, then slowly building up the structure by introducing these new concepts in the broader workflow, to see what impact it has on it as a whole.

Acknowledgements

I am usually not the best when it comes to these sort of parts, but I have to acknowledge that this graduation thesis would have been impossible without a few people - first of all, my coordinators, who were incredibly nice with me throughout this entire process. Also to the co-reader for taking the time to help me out with very useful suggestions. I would also like to thank my parents, family and friends for always being by my side whenever it was most challenging for me. Lastly, I would like to thank my workmates, who were also very supportive and understood me every time.

Contents

1. Introduction	1
1.1. Background Information and Problem Definition	1
1.2. Objectives and research questions	3
1.3. Outline of the thesis	4
2. Theoretical background and related work	5
2.1. Map Generalization	5
2.2. Line Generalization	5
2.2.1. Douglass-Peucker	7
2.2.2. Reumann-Witkam and Visvalingam-Whyatt	7
2.2.3. Samsonov-Yakumova	9
2.3. Vario-Scale Maps	11
2.3.1. tGAP	11
2.3.2. Topological structure	13
2.3.3. Spatial storing solutions	15
3. Methodology	19
3.1. Implementation of the Samsonov-Yakimova orthogonal Line Simplification algorithm in isolation	19
3.1.1. Median Implementation	23
3.1.2. Diagonal Implementation	26
3.1.3. Shortcut Implementation	27
3.1.4. Topological inconsistencies with the median simplification and alternatives	27
3.2. Integration with the broader tGAP-system and topological aspects	30
3.2.1. Correctly detecting intersections with external elements	30
3.2.2. Improving the selection of the edges to be simplified	33
3.3. Integrating multiple algorithms in the tGAP generation	33
4. Aspects related to Implementation	35
4.1. General Software details	35
4.2. Datasets	36
4.2.1. Main Data Structures used in the General tGAP generation Algorithm	37
4.3. Line Simplification Implementation	39
4.3.1. Coordinating the simplification	39
4.3.2. Used Data Structures and pre-simplification classification	39
4.3.3. Defining operations and simplifying	41
4.4. Details referring to the tgap integration	43
5. Results and analysis	45
5.1. The effect of embedding a line generalization algorithm into the tGAP generation workflow	45

Contents

5.2. Comparison of results from the user perspective	48
5.2.1. How highly-angular structures evolve in the generalization process . .	48
5.2.2. Situations where SY may be less than ideal	49
5.3. Consequences of introducing line generalization solutions in the tGAP gener- ation workflow	51
5.3.1. Understanding the impact of different algorithms on the run-time . . .	52
6. Conclusions and Future Work	59
6.1. Answers to Sub-Questions	59
6.2. Future work	62
A. Reproducibility self-assessment	63
A.1. Marks for each of the criteria	63
B. Various Figures and UML diagrams	65
C. Various Algorithm - in Pseudocode	71

1. Introduction

1.1. Background Information and Problem Definition

Throughout history, maps have evolved alongside all other technological advancements, from the Imago Mundi (Figure 1.1, left) from around the 6th Century BC, created in stone, which showed the city of Babylon with the river Euphrates in the middle of it [Lewy and Lewy, 1942], all the way to the highly interactive and easily adaptive maps which we can nowadays access at our fingertips. Over the last couple of centuries, our world has been continuously going through a technology-assisted globalization process, which has led to a society more interconnected and inter-dependent than ever, thus making cartography (and map building) and remote sensing alongside of it, an ever more important tool for billions of people across the world. ("Without maps we would be 'spatially blind' " [Gartner, 2014]) (Figure 1.1, right)

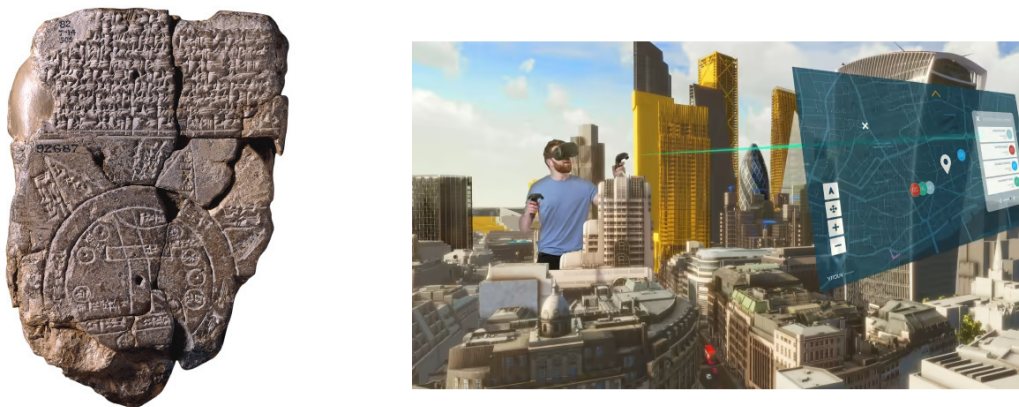


Figure 1.1.: Imago Mundi (left) [Williams, 2019] and its thousand year evolution, VR-maps (right) [Thomas, 2020]

When transitioning towards the digital age, paper-based maps have started to slowly be replaced by digital maps [Grand View Research, 2020], which also means that the way we interacted with maps had now changed drastically. One significant change from this development is the ability of map-users to now *switch in-between multiple scale levels*, thus allowing them to see more or less of the Earth's surface at one time.

However, at its origin, the map data, which has been collected by on-field surveyors, was suitable for a particular scale (usually, one would call this as the highest level of detail). From this point, it was traditionally the role of a cartographer to apply a map generalization process, at its own discretion [Punt and Conley, 2014], where we define generalization as "the process

1. Introduction

that simplifies the representation of geographical data to produce a map at a certain scale with a defined and readable legend (by applying a number of different processes). During the operation, the information is globally simplified but stays readable and understandable.” [Ruas, 2008] (further discussion on generalization to be found in the [Theoretical background and related work](#) Chapter) This task has shifted as well, from humans towards computers, which in term allowed a much more efficient process (for example, instead of a cartographer or a team of specialists working together towards converting from one scale to another, it is now possible to task a computer with this challenge.

The solution for storing the data, in all of its generalized forms, per different scale levels, which was adopted by most of the current market-leaders in web-based mapping solutions, is called “MULTI-SCALING”, and it refers to the process of storing information related to the map at discrete numbers of predefined scale levels (i.e. in a multi-scale database) - where the concept of scale can be understood, from the perspective of computing, as a measure of the distance on the field inside a certain pixel. Google maps is a notable example of a well-known provider using this technology. They store the data in a pyramidal format (Figure 1.2), where each level represents a certain scale value (in the case of Google, there are 19 levels), and where each higher level is composed out of an amalgamation of the pixels in the previous level [Google Developers, 2021].

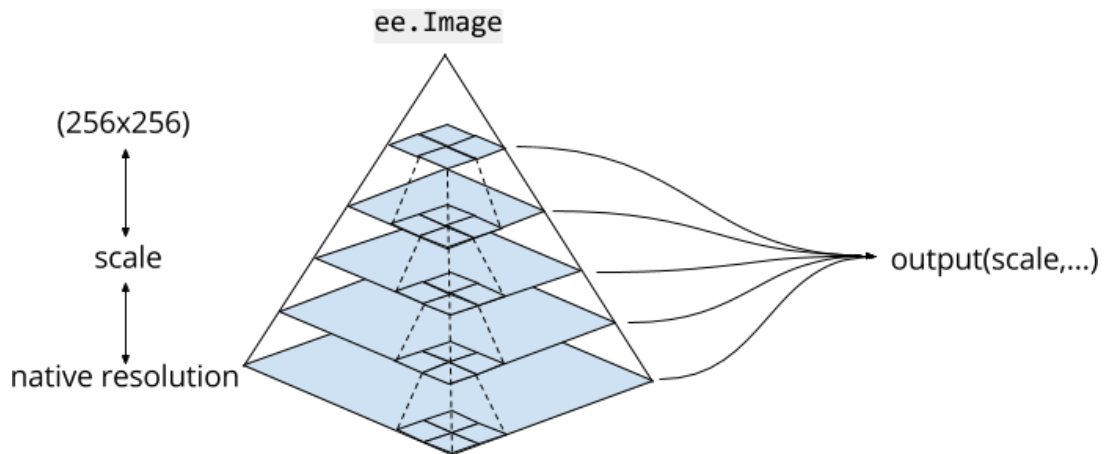


Figure 1.2.: A graphic representation of an image dataset in Earth Engine [Google Developers, 2021]

However, there are a number of issues which can arise when generating, storing and representing data in a “multi-scale” fashion: from one side, it may be argued that a certain amount of information could be lost this way, as it leaves a large gap of scales uncovered. At the same time, when storing data in discrete steps, the risk of data redundancy can arise, as the differences between one scale and the next one may not be so significant. On the other hand, there is also the issue of how map data is generalized: when shifting the task from a human to a computer, it comes as no surprise that sometimes elements on a map may end up looking strangely (counter-intuitively to how we would represent these features), while at the same time keeping the data too detailed, thus making it heavy to transport and store. [Dumont et al., 2020].

While the data storage and redundancy concerns are fixed with the implementation of Vario-Scale maps [van Oosterom, 2005], putting the various generalization decisions in the hands

1.2. Objectives and research questions

of a computer is far from trivial. While we end up gaining a lot of speed and efficiency when it comes to using an automatic generalization software, there were plenty of situations where the intuition of a cartographer would have come in handy when deciding how a particular element should be simplified, such that it ends up looking good from the human perspective.

Borders are a prime example of a difficult challenge that has yet to be solved. Of course, it is the desire of the user to see correctly-represented shapes on a map. At the same time, one needs to keep in mind the medium which is used to interact, which needs to handle a right amount of data such that the system can handle the task of displaying all of the features. Considering these requirements, the problem ends up becoming less and less trivial. Finding the perfect balance between preserving a good amount of data (with respect to the platform on which the map was intended to be used on as well as the overall scale and area which is currently being displayed), while at the same time preserving the overall characteristics of a border when transitioning towards a lower scale level.

Trying to automate the cartographic generalization process has been a continuous endeavour, and plenty of researchers and cartographers have tried finding the best solution. However, while a complete and final resolution of this problem may not be achievable, due to the many parameters, questions marks and the overall uncertainty surrounding both the map itself as well as its user, trying to find the best middle-ground, or a "best case scenario" solution, may still be an achievable undertaking.

Considering this brief introduction into the world of modern-day maps, alongside the insight into some the problems that these sort of mapping software solutions still encounter, it is time to start diving deeper towards the root cause and the issue itself. The following section goes over a subset of questions (out of a wider range of topics and issues all under the umbrella of the current subject matter), which will constitute the main research focus of this graduation thesis.

1.2. Objectives and research questions

After introducing the topic at hand, it is time to further restrict the subject matter by introducing more precise topics to analyse further in the graduation thesis. The spotlight will be put on the issue of borders, discussed in the previous section, and their generalization in particular, as it is one of the most important parts when it comes to having a good-looking map. To better encompass the main concepts to be discussed and investigated in this thesis, the following research question is introduced:

To what extent can multiple line-generalization algorithms be (simultaneously) introduced in the Vario-Scale structure such that they preserve the topology and enable an optimal line density (while trying to preserve the characteristics of the initial shape as well).

While this question does manage to bring the topic down to a narrower subject, it still has some degree of complexity. In order to better understand the research, and to go further in-depth into the problem, it would be a good idea to split the question into some more sub-questions. These will be divided into two categories, referring to theoretical aspects of line generalization as well as understanding how such algorithms can be implemented in an efficient way:

- Line generalization theoretical

1. Introduction

- Which line generalization algorithms are better suited for which particular situations?
- What is the most suitable way of combining said algorithms such that it upholds the technical requirements?
- Algorithm and Data structure design-related issues
 - What are the conditions and the development requirements necessary for maintaining topological correctness at any scale?
 - What is the optimal way of performing operations such that the line/vertices density remains constant, also when taking into account the scale change and its most favorable ratio between the number of objects and the size of the map which is being displayed at that particular scale?
 - How can the scale transition be performed in a smooth manner when integrating it into the broader Vario-Scale system? At the same time, what is the best way, from the point of view of time and size complexity (from the perspective of Big O notation concepts [Kuredjian, 2017], when looking conceptually at the efficiency of the various algorithms), to perform line generalization in particular and Vario-Scale operations in general?

1.3. Outline of the thesis

This graduation thesis begins with a presentation of the theoretical background in Chapter 2, which will serve as a starting point for the rest of the thesis. Afterwards, the main Methodology steps are introduced in Chapter 3, which give us the blueprint on which the rest of the thesis is built upon (and it may also be considered as a step by step guide on how to reach a solution). Next, a discussion on the main development concepts are described in Chapter 4, alongside other software and dataset constraints and requirements. Upon doing all of this development work, the following Chapter 5 presents the results, from both a visual as well as a numerical perspective, of the research which has been performed throughout this graduation process. Lastly, the paper ends with trying to answer the questions which were posed at the start of the entire process, in Chapter 6, which also tries to go over exactly how impactful the work done on this research has been on the overall scientific understanding of this topic.

2. Theoretical background and related work

This chapter serves as a theoretical starting point of this thesis, and aims to introduce the main building blocks behind the research. It begins with giving the big picture of the concepts behind the map generalization process, followed by a more in-depth discussion into the sub-category of border generalization. After these concepts have been introduced, we can take a deeper-dive into the inner-workings of the Vario-Scale systems, including what it is, what structures it uses and how the underlying data should be modelled and structured. Lastly, a number of different spatially-aware data structures are described.

2.1. Map Generalization

Müller and Wang [1992] defines **Cartographic Generalization** as the process of abstracting, in a meaningful way, the diversity and complexity of the real world such that the resulting cartographic representation is useful and usable with respect to the given scale and overall purpose of the map. There are a number of distinct **basic** processes which can be applied in the generalization workflow (Figure 2.1), out of which only the (line) simplification and the amalgamation are used in the Vario-scale map generation (as well as a split operation, not depicted in the guide by Shea and McMaster [1989], which splits a face into pieces and then it assigns those pieces to the neighbours, further explained in Section 2.3.1 [van Oosterom and Meijers, 2014]). Although there is a lot of theory related to cartographic generalization, for the purposes of this thesis, it would be good enough to just discuss about the line generalization process.

2.2. Line Generalization

Line Generalization is the process of simplifying the shape of polylines (representing the borders of object on a map), such that the overall result has a shape more suited for the scale at which the operation is performed. There are two main procedures of going about this simplification, namely:

- line simplification, which removes, according to a certain set of rules, a varying number of points from a targeted line.
- In Shea and McMaster [1989] line smoothing is defined as “act on a line by relocating or shifting coordinate pairs in an attempt to plane away small perturbations and capture only the most significant trends of the line”. Thus, this operation only changes

2. Theoretical background and related work

Spatial and Attribute Transformations (Generalization Operators)	Representation in the Original Map	Representation in the Generalized Map	
	At Scale of the Original Map	At 50% Scale	
Simplification			
Smoothing			
Aggregation			
Amalgamation			
Merge			
Collapse			
Refinement			
Typification			
Exaggeration			
Enhancement			
Displacement			
Classification	1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20	1-5, 6-10, 11-15, 16-20	Not Applicable

Figure 2.1.: Different types of generalization solutions[Shea and McMaster, 1989]

the position of certain nodes in the polyline, in incremental steps, until the two segments which converged on that node become co-linear, thus creating a smooth border animation when going in-between different zoom levels¹.

Notable examples of line smoothing algorithms include McMaster's Distance Weighting Algorithm [McMaster and Shea, 1992] or Chaikin's Smoothing Algorithm Chaikin [1974]. However, while this operation may be a very useful in certain situations, it ends up creating extra complexity by changing the structure of the map without actually simplifying it. Thus, smoothing is not explored any further in this research, but it is mentioned later in the discussion of [Conclusions and Future Work](#).

While line smoothing solution can provide visually pleasing map for the end-user, it does not involve the removal of features, and thus does not make the map any simpler. As using these particular algorithms does not completely fall within the scope of this thesis, we will only be looking at the "line simplification" category, in the rest of this section.

¹<https://geogra.uah.es/patxi/gisweb/LGmodule/LGSmoothing.htm>

2.2.1. Douglass-Peucker

While this solution is not used directly in the thesis, it does deserve a mention, as it is used in the BLG-tree in the original version of tGAP, and it is also the first such solution, dating from 1973 [DOUGLAS and PEUCKER, 1973]. At the same time, it is still being used in various pieces of software, including Esri's own "Simplify Line" toolset², where it is considered suitable for retaining critical points, and even in applications such as GPS and Waypoint tracking, or AI [Mohneesh, 2021]. For these reasons, it can be considered as a very valid reference point when using it to compare against other similar line generalization algorithms.

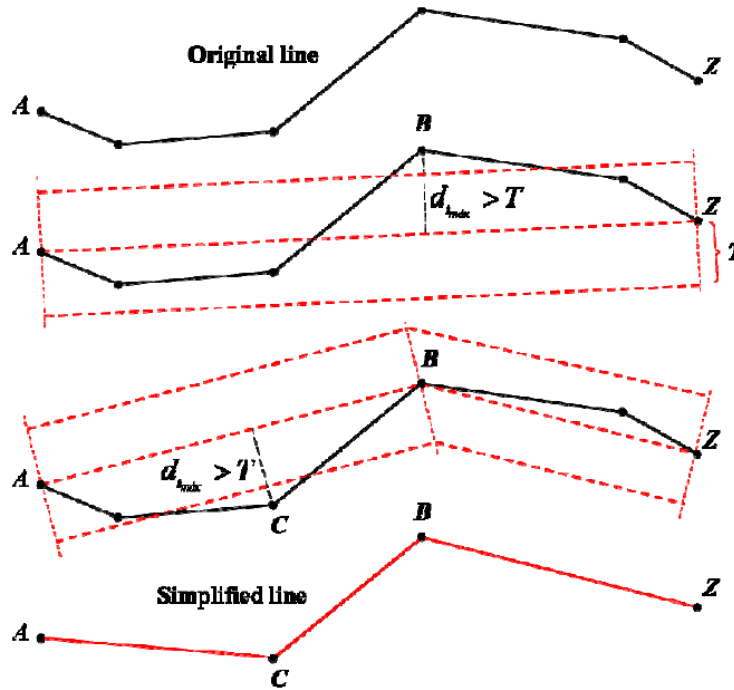


Figure 2.2.: DP Simplification [Crespo et al., 2008]

The Douglas-Peucker line simplification is an iterative algorithm, which uses a predefined tolerance value to check against all the component points in a particular edge. It starts by connecting the start and end point, then adding to the furthest away point from the baseline to the list of saved points (second step in Figure 2.2). This process occurs in an iterative manner until either all points are outside the threshold (Step 3 in Figure 2.2), while all nodes which are inside of the threshold will end up being removed.

2.2.2. Reumann-Witkam and Visvalingam-Whyatt

Both the Reumann-Witkam as well as the Visvalingam and Whyatt [1992] algorithms come as alternatives to Douglas-Peucker, as it has a number of edge-cases where its result is less than ideal.

²<https://desktop.arcgis.com/en/arcmap/10.3/tools/cartography-toolbox/simplify-line.htm>

2. Theoretical background and related work

The Reumann-Witkam algorithm, similarly to the Douglas-Peucker alternative, uses a tolerance value to determine which points should remain in the simplified version. That being said, the way the tolerance is used is differently³. Instead of firstly taking the start and end points of the original geometry, the iteration is performed by taking every two consecutive points. Then, any other vertices which lie within the tolerance value are removed (as it happens in Figure 2.3). This is being repeated until we reach the second-to-last and last vertices respectively.

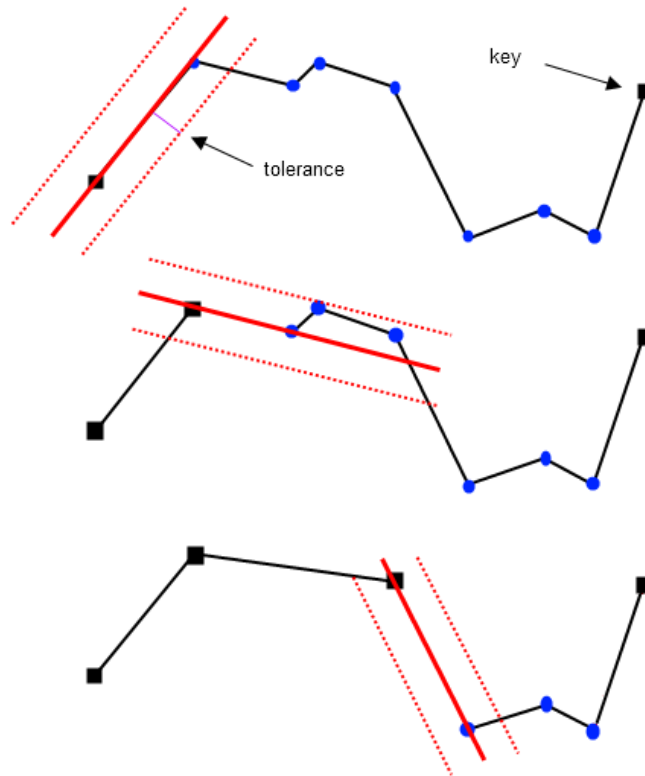


Figure 2.3.: The Reumann-Witkam Simplification - visual explanation [Source - psimpl website]

On the other hand, Visvalingam and Whyatt approach the solution in another way: instead of using a tolerance, they take all triangles in the polyline (formed by every three consecutive vertices). Then, the triangle with the smallest area is selected for removal, which results in it being flattened (basically, for the solution, the base of the triangle is connected - as it can be seen in Figure 2.4). This process is being repeated iteratively until only the first and last nodes (from the initial structure) remain.

³<https://psimpl.sourceforge.net/reumann-witkam.html>

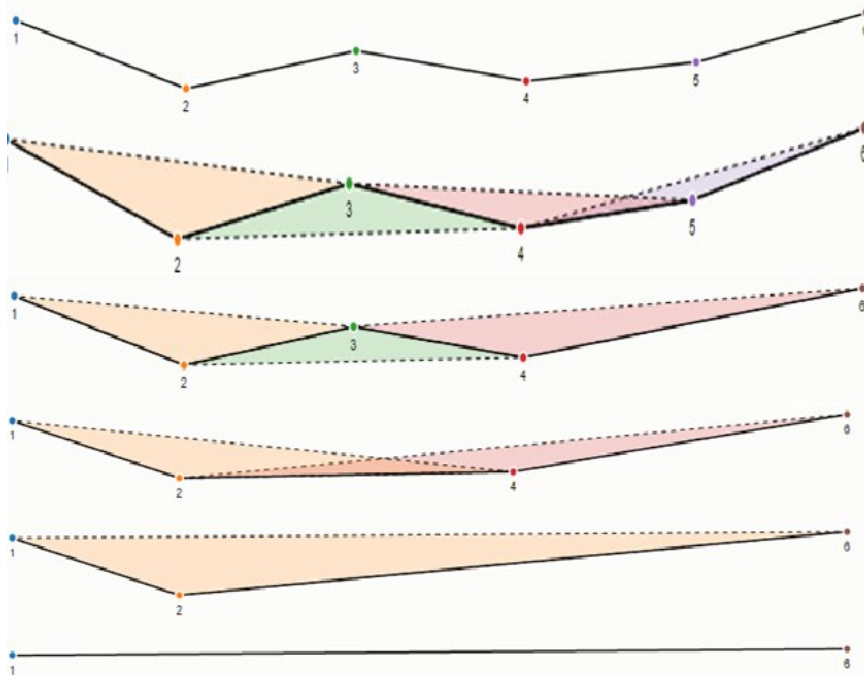


Figure 2.4.: VW Simplification [Source Amigo et al. [2021]]

2.2.3. Samsonov-Yakumova

The Samsonov-Yakimova “Shape-adaptive geometric simplification of heterogeneous line datasets” paper introduces a complex algorithm which deals with complex geographic data. Unlike the previously mentioned line generalization solutions, the two researchers focused on incorporating a number of different algorithms, depending on the type of border which exist in a particular dataset. Looking at the classification in Figure 2.5, the authors consider that common administrative divisions, which generally follow either parallel or median directions, as being perceived in an orthogonal and schematic pattern. Oppositely, divisions created by natural geographic elements, such as rivers or mountains, will follow a non-schematic (smooth) classification, without any inherent regularity, and will require in term a different kind of simplification.

For this thesis, only the algorithm which is to be applied to highly-schematic and regular borders will be of relevance. For this reason, a number of steps (such as pre-processing, or classification) will be skipped in this explanation. And, even though the solution presented is used to discuss highly-artificial administrative borders, the algorithm itself is very useful for man-made structures as well, such as buildings.

In a nutshell, the paper discusses the need of preserving the angularity of these schematic-orthogonal lines, in such a way that their squared characteristic (i.e. 90 degrees angles) is kept even after the simplification. In the original version of the solution, a target scale is being set (by the user), where any segment shorter than the length-tolerance will be removed

In order to determine which of the simplification should be used, Samsonov and Yakimova [2017] derived a configuration, where the borders were classified in either a Z-like, U-like,

2. Theoretical background and related work

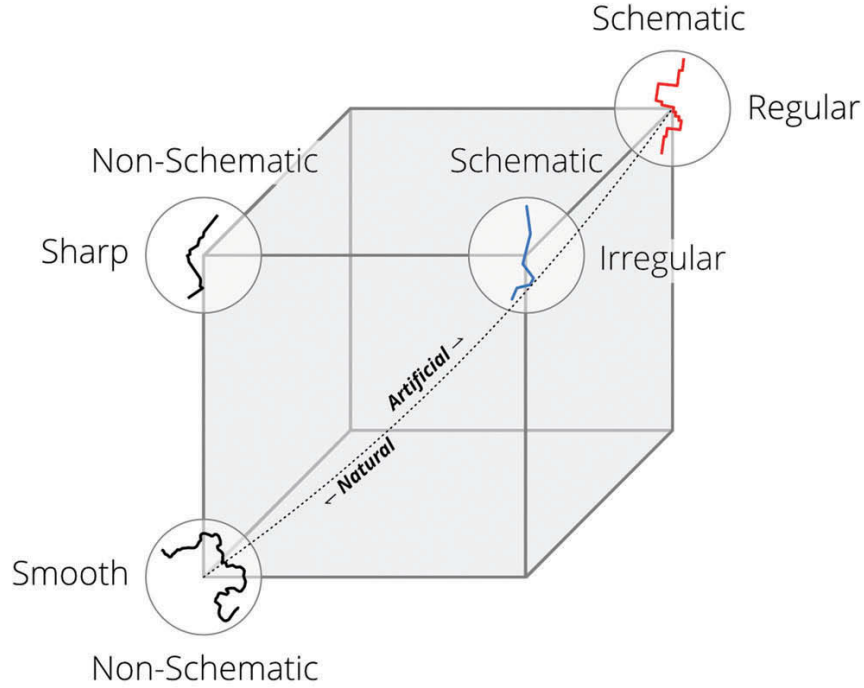


Figure 2.5.: Classification of border types [Samsonov and Yakimova, 2017]

endpoint or short configuration (Figure 2.6). The simplification itself is based on “substitution procedure in which the deleted line segment and its neighbours are replaced by a new configuration”, where 3 types of substitutions are included in the algorithm itself, namely the median, shortcut and diagonal (Figure 2.6, top-right).

The median simplification is performed by removing the “problematic” segment, e_i , alongside its direct neighbours, afterwards being replaced by a perpendicular line from its second-degree neighbours ($e_i - 2$ and $e_i + 2$ respectively). As this process end up equalizing the overall proportions between two neighbouring faces (and making the pattern, in the vision of the authors, more perpendicular), it is considered more suitable for small-scale generalization, thus making it the default option for both the U-like and the Z-like configuration.

Shortcut substitution, on the other hand, works by creating a connection between the $e_i - 1$ and $e_i + 2$, or alternatively, $e_i - 2$ and $e_i + 1$. The paper suggests applying both algorithms, and selecting the one which does not introduce any topological errors.

Lastly, the Diagonal substitution is applied in cases where the edge consists of three interior nodes (or an edge which has exactly four segments), and it implies simply connecting the first and the last interior point. At some point in the generalization process (when the scale becomes extremely low), all segments will end up applying this substitution method.

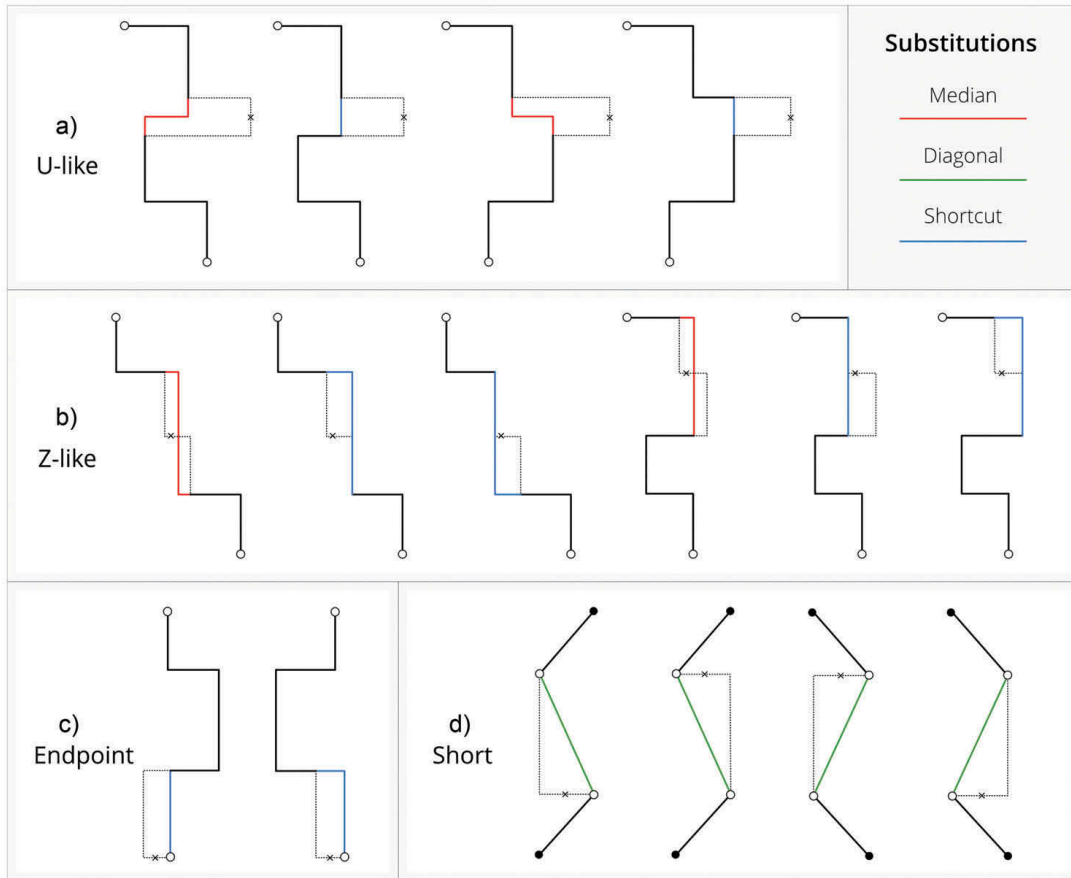


Figure 2.6.: “Edge substitution strategies for various configurations of orthogonal line segments. Endpoints are symbolized by hollow dots, the old configuration is a dotted line, the contracted edge is marked by an X and the new configuration is depicted in colour.” [Samsonov and Yakimova, 2017]

2.3. Vario-Scale Maps

2.3.1. tGAP

The topological Generalised Area Partition (shortened henceforth as tGAP), introduced in the paper by van Oosterom [2005], is a data structure suitable for storing the sequence of operations which are being performed in the Vario-Scale generalization process, without any geometrical redundancies. It comes as an alternative to the data structures used in Multi-Scale mapping architectures, which have the drawback of storing duplicate data across multiple scale levels (in the situation of geographic map objects which undergo little to no transformation in-between two (or multiple) scales, it would be required to keep the same record in the datastores associated to the respective scale level), while at the same time only allowing for a *limited* number of available Level of Detail.

The tGAP comes as an improvement to the original GAP-tree [van Oosterom, 1995], and its

2. Theoretical background and related work

creation is based on two main functions: an Importance equation (Eq. 2.1), which gives an importance value to all faces based on their area and the weighted value of their respective classification. The second function (Eq. 2.2) computes a comparability parameter between two adjacent polygons based on the length of their shared border and a pre-defined value which represents how compatible two different classes are.

$$Importance(a) = Area(a) * WeightClass(a) \quad (2.1)$$

$$Collapse(a, b) = Length(a, b) * CompatibleClass(a, b), \quad (2.2)$$

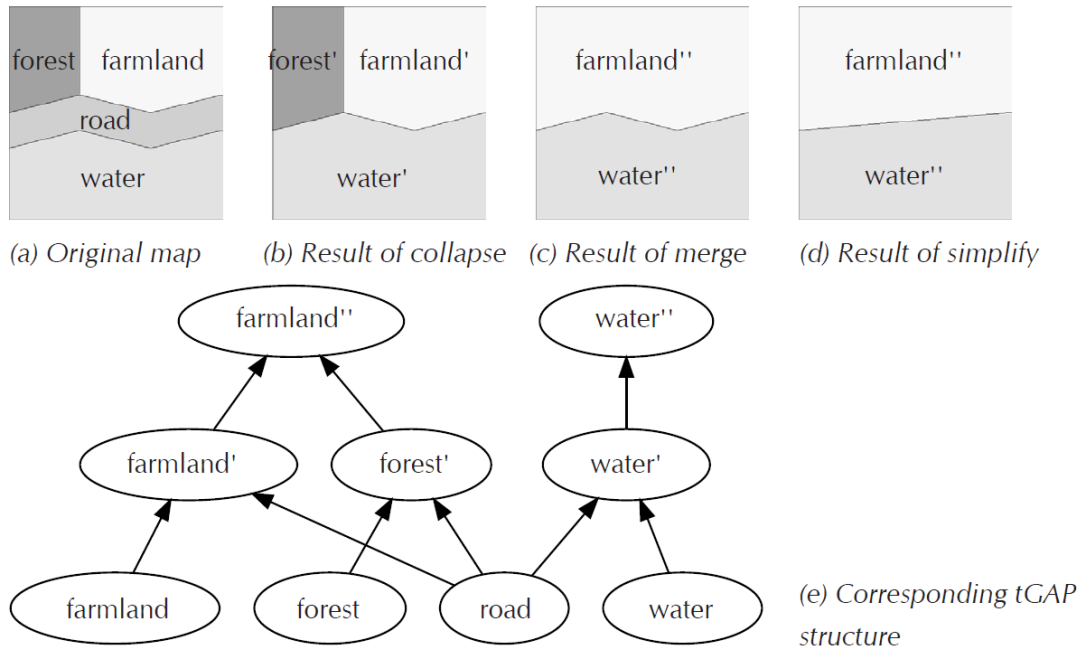


Figure 2.7.: The 4 map fragments and corresponding tGAP structure [Source: [van Oosterom and Meijers \[2014\]](#)]

Using these two equations, it is then possible to determine a list of n elements containing the least important areas, alongside their most compatible neighbours, thus determining the area which will be assimilated into which adjacent polygon. This is also called the merge operation, and is depicted in Figure 2.7, at step 'c'. This process would happen until there is only one single object left.

After giving an introduction into the equations which formed the merge operation in the initial GAP tree structure, [van Oosterom \[2005\]](#) goes over a number of improvements which occurred from that point. One new (software) addition worth mentioning would be the introduction of the split operation in [Ai and van Oosterom \[2002\]](#), based on the idea that, instead of finding determining which of the neighbours is most compatible with a certain area which has to be removed, we could split the area amongst all of its neighbours.

However, it is pointed out in the paper that these subsequent versions of the GAP tree still did not contain any topological information. Thus, a topological structure is attached to the

GAP tree, meaning that "the edges and the faces table [in the output of the tGAP generation] both have attributes that specify the importance ranges in which a given instance is valid".

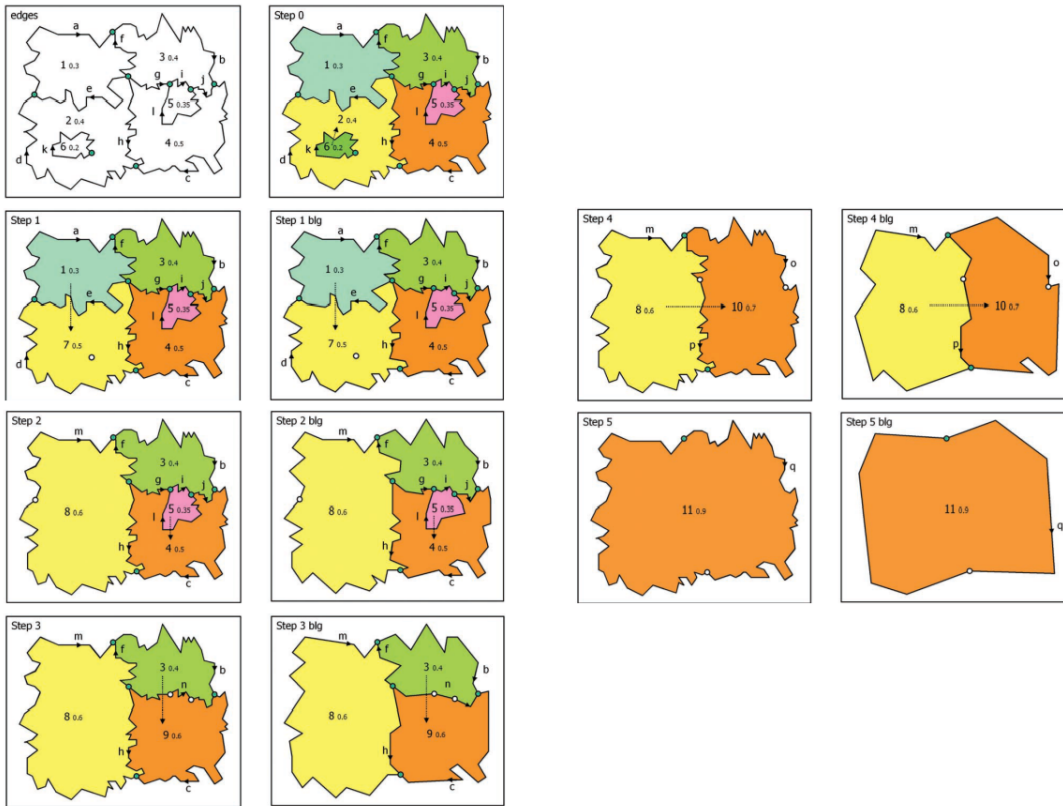


Figure 2.8.: The step by step process (from the perspective of the objects on the map) of the tGAP Generation Workflow [Source: [van Oosterom \[2005\]](#)]

A line simplification operation, alongside the merge and split functions, was being handled by a BLG-tree. This uses the Douglas-Peucker algorithm (Section 2.2.1) to simplify the edges and to record these performed operations in a tree structure while also preserving information related to the change which has occurred. An example on how these operations are performed is presented in Figure 2.9

The focus of this graduation research will be exclusively on the tGAP generation algorithm. The second part of the Vario-Scale workflow, which deals more with visualization, is called the Space-Scale Cube (further also abbreviated as SSC). As it is not researched in this paper, being aware if its existence is more than enough at this point. However, it would be worth mentioning that, the topic presented in the next sub-chapter, while only discussed in the frame of the tGAP generation, has effects on the SSC as well (Example in Figure 2.9)

2.3.2. Topological structure

Considering a particular planar partition as the starting point (depicting a certain finite area of the earth, with more or less abstractions pre-applied to it), we can refer to and define a num-

2. Theoretical background and related work

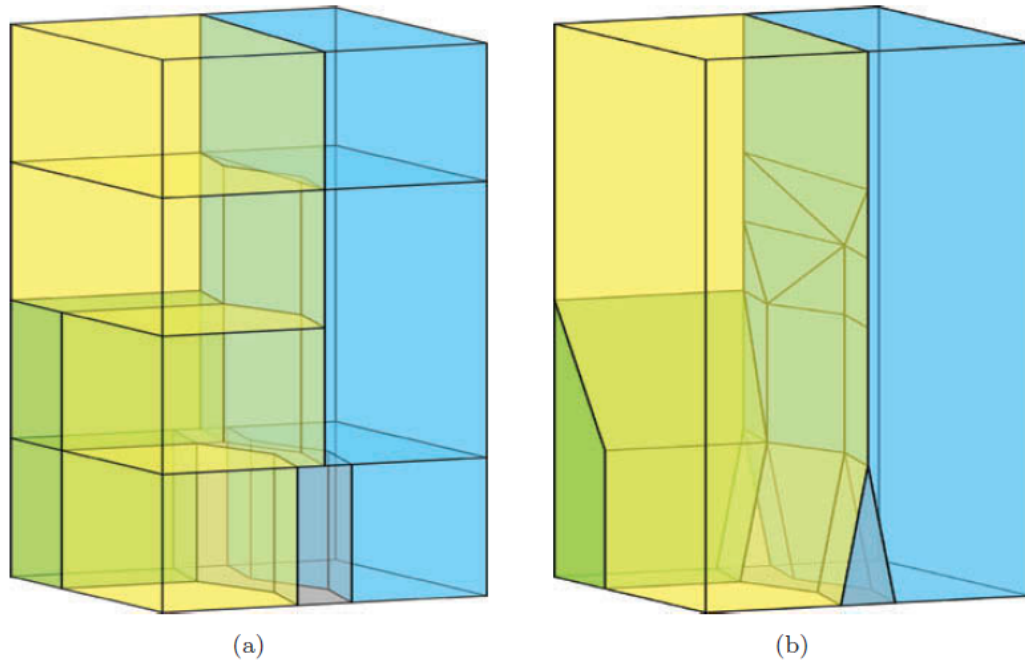


Figure 2.9.: An example of a Space Scale Cube, when using a) the classic version of tGAP and b) the smooth version [Source: van Oosterom and Meijers [2014]]

ber of different cartographic generalization operations which can be applied on the planar partition, in a sequential order, until there are no more actions which can be done (meaning that the planar partition will contain only *one, fully-simplified* polygonal object/face).

When working with a vario-scale system, we take, as input data, a topologically-correct planar partition, which will represent the starting point for the generation of the vario-scale map. This is also considered as the highest level of detail. A planar partition is defined as “a subdivision of a polygonal subset of the plane into non-overlapping polygons” (which are also referred to as faces) [Ohori et al., 2012]. These faces are in term composed out of a number of polylines, which also constitute the borders between any two polygons.

Before going into a deeper discussion regarding this topic, it should be noted that the aforementioned planar partition should contain information related to what kind of real-world classification do the polygonal faces adhere to. This enriches the dataset, thus transforming it from a simple representation of some geometrical objects (points, lines, polygons) on the 2D plane into a spatially (and topographic) collection of data. For example, a forestry GIS specialist may receive a dataset from land-surveyors containing a plain planar partition composed by purely-geometrical instances (which may or may not be referenced in a certain Coordinate Reference System). However, categorizing each polygon as a certain species of tree would significantly improve the usefulness of the data.

This classification can be then further refined by assigning a certain weight to each of the classes based on how important that respective land-use classification is in a particular instance (for a certain application) or based on the preference of either the user or the producer

of the Vario-Scale map (For example, a governmental agency which deals with infrastructure projects will of course give a higher priority to road/train/canal network rather than forested areas, which will mean the overall generalization process (and in particular the tGAP generation) will favor infrastructure-like polygons to their other counterparts in different decisions).

An in-depth discussion into this topic is presented in the paper by [Ohori et al., 2012], including a presentation of the steps taken in preparing the dataset, such that it adheres to all topological constraints that are required for the process to work. In the scope of this graduation paper, it would be good to briefly mention (some of) the more important criteria, which would ensure the topological correctness of any data structure (in the shape of a planar partition, when working in a tGAP-bound system):

- *Completeness* - The entirety of the area defined by the planar partition should be covered completely by non-void objects. What this means is that, considering we have a vector map depicting a particular area, there should not exist any blank spots, or locations with no faces present. The only non-void face, also known as the outside face, is only allowed to exist in the exteriors of the boundary of the planar partition.
- *Correctness* - No two geometries should intersect with one-another, and no line segments may have overlaps/self-intersections. Considering that one edge has exactly one left and one right neighbour, either of these situations would in term create the case where there are conflicting faces being considered for the same-side neighbour.
- *One node multiple edges validation* - Lastly, and as an addition to the previous point, intersections in a planar partition are only allowed in geometric structure called 'Nodes', which are defined as 1D locations, where more than two strictly distinct edges may end-up in (a definition which clearly excludes both self- and outside-intersections, as they would imply the same edge getting out of the Node more than just once). Any Nodes with exactly two edges going out of them (either the same edge, or a different one) should be dissolved, and one single edge should be put in its place, to keep this requirement.

Skipping over any of these requirements will most likely result in an the workflow malfunctioning from its usual behaviour, due to the fact that the different generalization processes will end up giving an erroneous result. This includes the various line generalization algorithms which are used throughout this paper. Thus, once it can be confirmed that the dataset is correct, we can move on by looking at the aforementioned line generalization solutions, and to understand the theory behind their inner-workings.

2.3.3. Spatial storing solutions

As we are working with very complex data, which can have potentially Gigabytes or even Terabytes of data points, having a way to efficiently store and access it is crucial. For this reason, having a good data structure can truly make the difference between an efficient and a slow solution.

Quadtrees are, as the name suggests, a tree-based data structure, which work by recursively sub-dividing a geographic space, filled with elements, in a particular manner, such that the final resulting tree will allow easy access of the component elements which are close to one another, thus enabling very efficient Nearest neighbour operations, or even various geo-distance

2. Theoretical background and related work

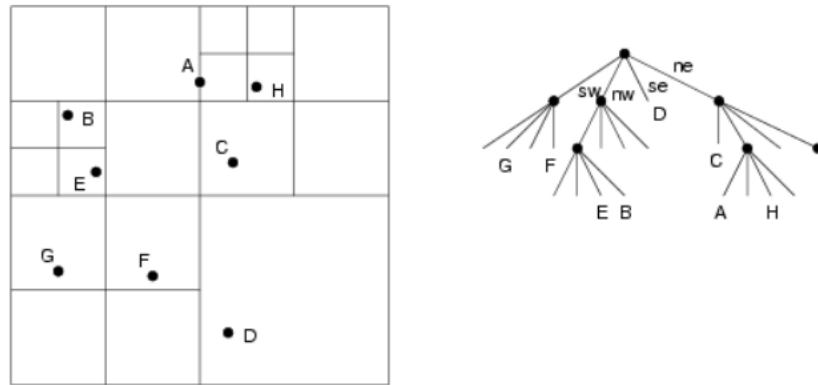


Figure 2.10.: An example of a PR Quadtree [van Oosterom, 2009]

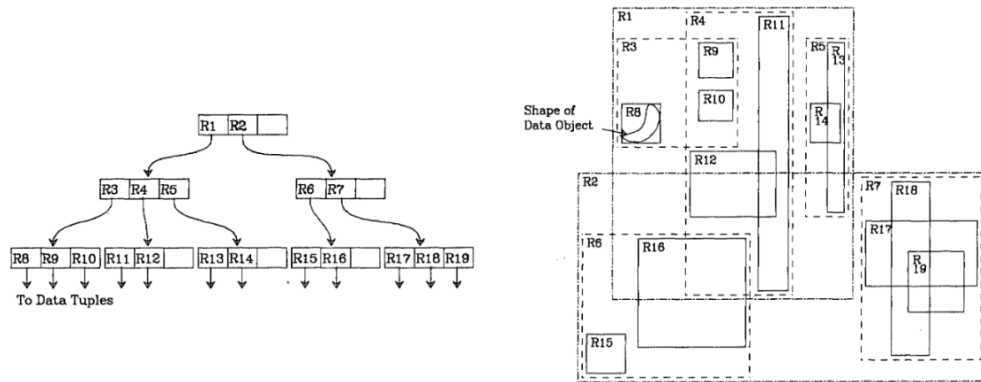


Figure 2.11.: Example of R-tree [Guttman, 1984]

range queries [Oyediran, 2017]. The term of “quadtree” is a general one, and it incorporates a number of different variants.

Point-region-quadtrees (PR-quadtrees) is one such category, where the space is divided into four different (but equal) quadrants, in accordance to the four geographical locations: SW (south-west), NW (north-west), SE (south-east), and NE (north-east) [van Oosterom, 2009]. At the end of the division, each bucket should contain at most one point (Figure 2.10).

As its name suggests, it is a very useful data structure when it comes to storing and accessing spatially-distributed points (defined as (X,Y) coordinates on a plane). However, when it comes to more complex types of data, such as simple or complex edges, an alternative such as R-trees may prove to be a better solution.

R-trees were created by Guttman [1984] with the scope of handling geometrical data as efficiently as possible. It works well not only with points, but with lines, rectangles and other polygonal structures as well. The R-tree works by bounding any object which exists in a data set within a Minimum Bounding Box, which will be leafs of the tree. Further-up, these Boxes

are further grouped into bigger and bigger regions (which will end up being the nodes of the tree) [Papadopoulos et al., 2009]. Exactly how these groupings occur can be adapted through the algorithm itself, and a structure will end up looking similar to Figure 2.11

The usefulness of these data structures, when working with a Vario-Scale system in particular, lies in the ease of data access, especially when it comes to the generalization process. Due to the large number of simplifications which are being performed during this process, having an easy way to check any neighbouring features in the region where we are working can truly make the difference (for example, in the case of intersections)

3. Methodology

After establishing a solid theoretical background in the previous chapter, it is time to take an in-depth look at the methodology applied in investigating the research questions which were proposed. Only the conceptual ideas are presented in this part, the actual implementation of the solution is discussed in the following chapter.

The general workflow is shown in Figure 3.1, and it gives an overview of the four main stages in the development of the Methodology. These stages may also be read and interpreted as steps taken in order to come to a definitive solution. The first step discusses the main design choices behind the implementation of the Samsonov-Yakimova Line Simplification algorithm, taken from the perspective of isolated geometries. In the next layer, we take the resulting solution and plug it into the tGAP Generation workflow. Once it is confirmed that the new workflow functions correctly, the next step in the methodology is to combine the previously-developed orthogonal line simplification with the already-existing solutions (such as Reumann-Witkam or Visvalingam-Whyatt). In the last step of the whole process, after the integration of multiple solutions has been successfully completed, a discussion on further improving the synchronisation between the processes can be had. Finally, the output of the entire methodology should be an improved version of the original solution for the tGAP Generation process, where more than just one line generalization process can function in harmony both amongst themselves as well as with the other generalization operations which are used. From a system development perspective, the aforementioned steps may also be considered as parts in a Workflow Diagram¹.

Before going further into the Methodology itself, for the purposes of later on answering the sub-questions defined in the Introduction chapter, it is good to define exactly what it means for an algorithm to adhere to "technical requirements". We can consider "technical requirements" as a set of rules defined for classifying the workflow as 'having ran successfully'. In particular, these rules are: the solution runs without any errors; the solution should generate a correct result; and lastly the solution should consider the right algorithm for the right case. Considering this definition, and the rationale behind the process, it is time to take a further in-depth look at the steps performed to research the premise of this thesis.

3.1. Implementation of the Samsonov-Yakimova orthogonal Line Simplification algorithm in isolation

Considering the issues presented in the previous section, and before going further in-depth into how to choose the optimal line simplification solution under varying circumstances, at least another algorithm had to be implemented. Due to Reumann-Witkams' (or even Visvalingam-Whyatt's) algorithm less-than-desirable way of handling highly-geometric (or man-made) polygons, finding an alternative for this situation seemed like the logical next

¹<https://www.lucidchart.com/pages/tutorial/workflow-diagram>

3. Methodology

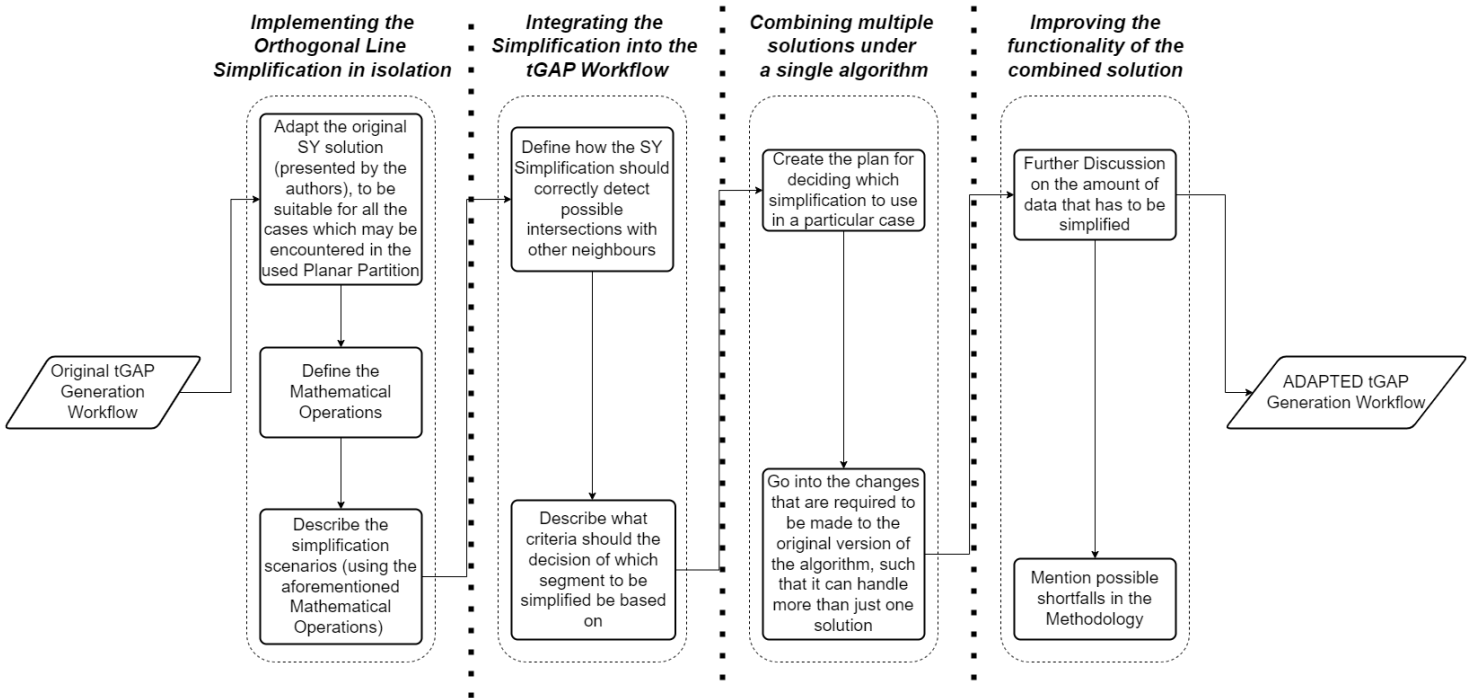


Figure 3.1.: The Methodology diagram

step forward in the research process. The algorithm presented in the paper by [Samsonov and Yakimova \[2017\]](#) for lines which they define as Sharp regular schematic/orthogonal (discussed in Section 2.2.3, classified in Figure 2.5) was chosen to perform this task.

While the original solution worked well for the dataset used by the authors, due to the complexity behind the way our data is structured, it had to be adapted such that it would work in harmony with the other operations present in the tGAP generation workflow. This means that there have been a number of tweaks done on the initial version of the algorithm, so that all situations and edge cases which may exist when working with real-world data are covered.

First of all, there was a requirement to adapt how data elements are used in the simplification operation - while Reumann-Witkam algorithm takes points as arguments, we use segments. Further information on this is given in the [Aspects related to Implementation](#) Chapter, but for now it is worth mentioning that the segment will constitute our smallest unit, on which modifications are applied. This means that, if for RW we were discussing about removing point from the polyline, we are now considering the removal of segments (Figure 3.2).

Another important difference, that has an effect on the the algorithm's way of reaching a final result, is the fact that the Samsonov-Yakimova line simplification, due to the way that it was defined, ends up introducing new Point geometries into the mix. This is not something that occurs in the case of RW or VW, as these two solutions only connect already existing nodes (exactly how this occurs differs for the two algorithms, but the process itself is more or less the same).

By conceptualizing the different scenarios presented in the paper by [Samsonov and Yakimova](#)

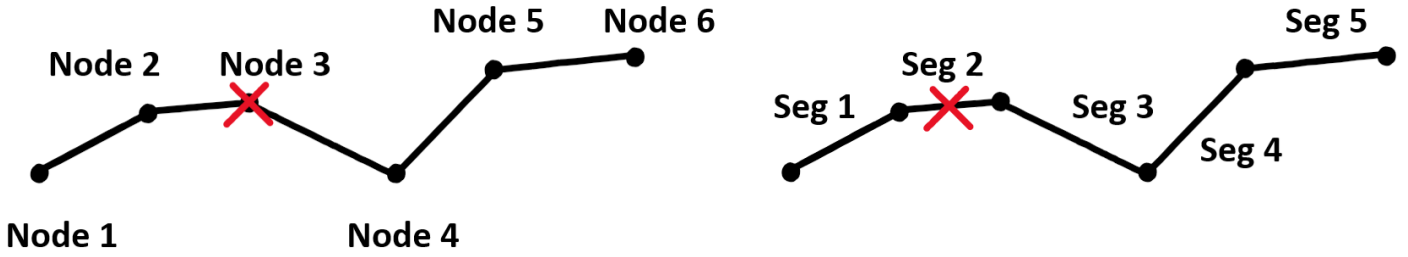


Figure 3.2.: How different basic unit of operations are considered in different situations (RW on the right, SY on the left), where the removal operation is being performed at a point and segment level respectively

[2017] (the U-like, Z-like, Endpoint and Short configurations and the Median, Diagonal and Shortcut substitutions) under the different situations which can occur when working with this sort of data (further insight on the data itself is given in Section 4.2 of the following chapter), a number of different versions can be devised, in which a certain type of operation is applied:

Considering a certain line containing n number of segments, which has a pre-defined order from first (start) to the last (end) division, and where $n \geq 4$, we describe a segment i as "undesirable" if it complies with a certain criterion, which then results in being chosen for elimination - in the case of this particular methodology, this criterion is the length of the segment, where the shortest one is selected for simplification. After this particular segment has been selected, we then define:

- Direct/First-degree neighbours, as those segments which are directly to the left and right of our "undesirable" i
- Second-degree neighbours are the segments directly to the left/right of the first-degree neighbours.
- An "anchor" point is the extremity of a certain segment (either its Start or its End) - which is considered to be relevant in the context of a particular operation – A "border" or "terminal-position" segment is either the first or the last segment in a non-circular polyline. Their immediate neighbours are called "neighbouring-the-border" or "almost-terminal" segments

We define a line to be "closed" or "circular" if its starting-point is the same (spatially speaking) as its end-point. A segment chosen on this sort of polyline will always have all direct and second-degree neighbours, while this is not always a guarantee for non-circular polylines.

Given these general definitions, we may now go ahead and fit different situations under similar cases, in Figure 3.3 (further explanation on the drawing symbols given below). Each of this case can fall under a different Simplification variant.

- Case A: This is the situation where the segment to be removed has all of its left and right first/second degree neighbours. This is the case under which all circular polylines fall. In this case, we can apply either the **Median Simplification** (Which is shown in the Figure) or the **Shortcut Simplification**
- Case B: The undesirable is one segment away from being the final one in the line (meaning its either the second or the second-to-last). This is the situation which is not covered by the authors of the algorithm. Due to the fact that one of its second degrees

3. Methodology

neighbours doesn't exist, the **Median Simplification** cannot be correctly applied, so the decision has been made to apply the **Shortcut Simplification** in this scenario

- Case C: this is the "endpoint" configuration presented in the paper. In this case, the segment to be removed is either the first or the last one in the polygon, and thus has no neighbours on one of the sides. **Shortcut Simplification** will be applied
- Case D: Equivalent with the "short" configuration, occurs only when the number of segments in the polyline is 4, and the one eliminated is in the interior (i.e. NOT the first or last one). The **Diagonal Simplification** is applied in this situation

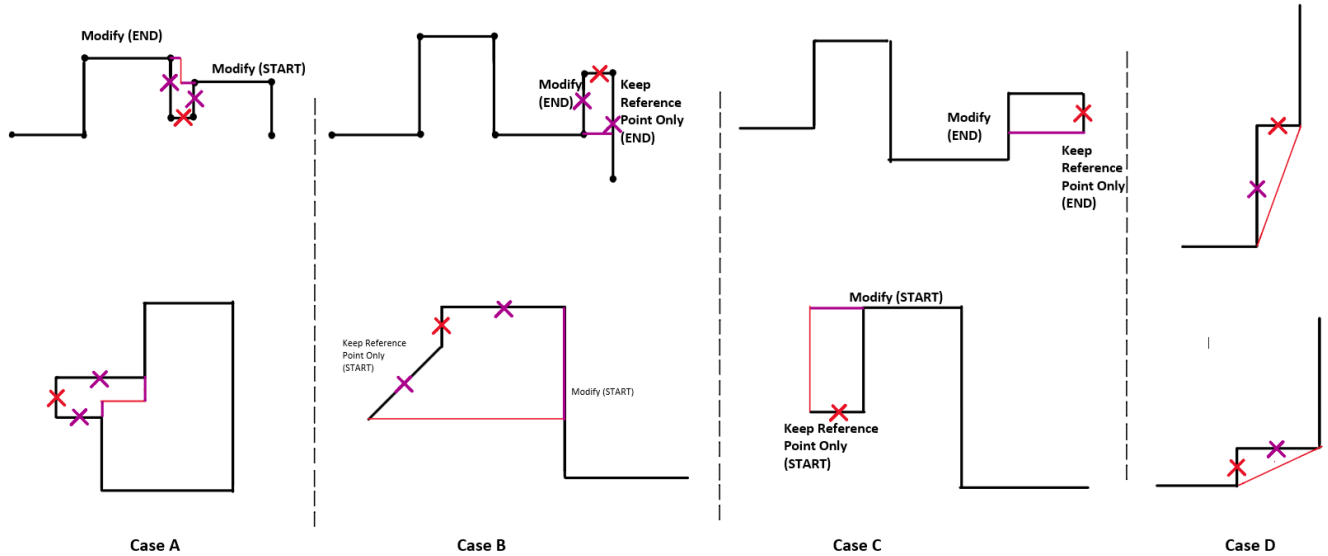


Figure 3.3.: The different scenarios which may occur during the orthogonal line simplification

In Figure 3.3 the segment which has been selected to be discarded is marked by a *red cross*, while the segments which are removed alongside the main one are noted by a *purple cross*. Segments which have been extended are *purple*, while newly-created ones are shown in *red* lines. On the same figure, the procedure which is applied to a certain segment is displayed (using an arrow), alongside its relevant/anchor point. Overall, there are 3 main operations which can be applied on a certain segment (Figure 3.4):

–*Complete removal*: this is where the segment (alongside both of its endpoints) are taken out of the polyline.

–*Modification*: In this procedure, we modify the location of the anchor point along the general direction of the segment (thus extending or contracting the segment length), until it intersects with a ternary line equation (purple). The other end will remain in place

–*Keep reference points only*: here, the anchor point is the one that remains in place (fixed), and from it a new segment (with a different line equation!) is created, by tracing the perpendicular from the anchor point to the ternary line (with purple)

In the situation of the latter two operations, there are two different directions (namely left and right) in which they can be performed. The decision on which of the two options to utilize will be based on the location of the particular segment to be removed in the context of the

3.1. Implementation of the Samsonov-Yakimova orthogonal Line Simplification algorithm in isolation

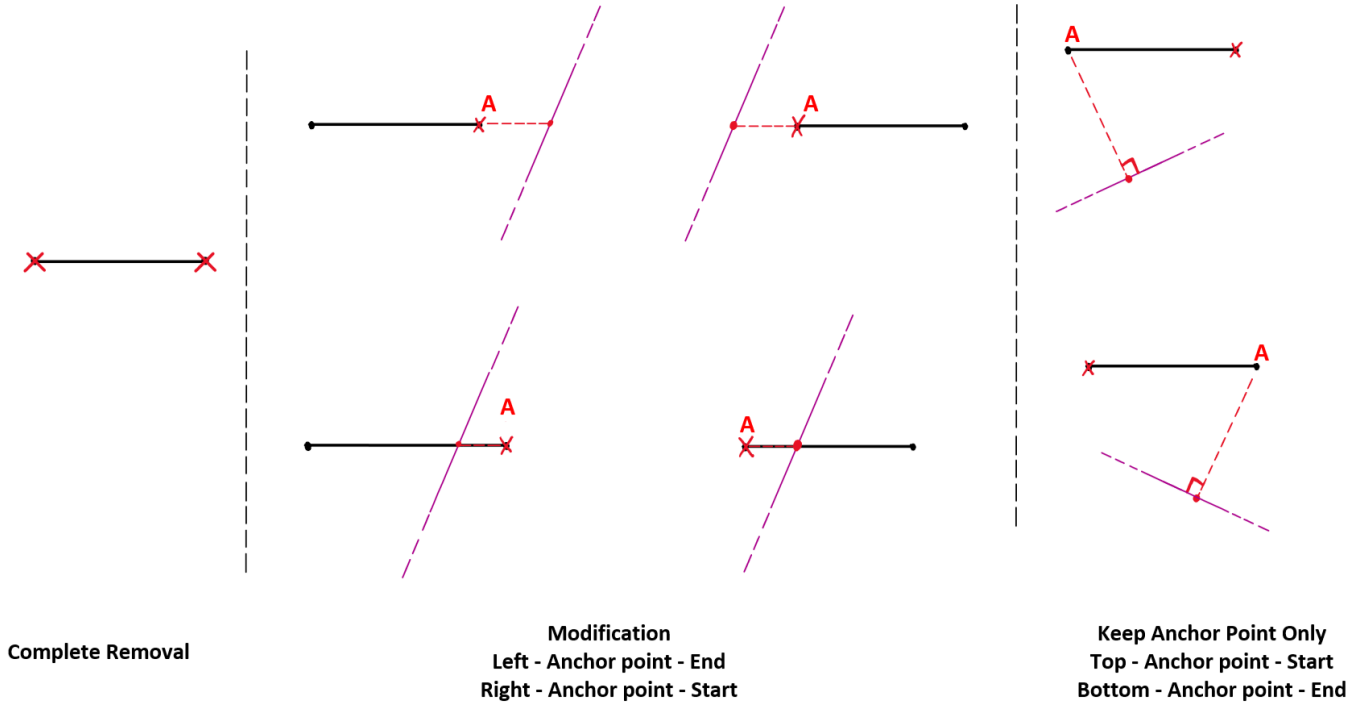


Figure 3.4.: Main actions which can be applied on a certain segment

edge which has been selected for simplification. While the mathematical formulas which will be applied will remain the same, the variables will change depending on the situation.

3.1.1. Median Implementation

In a Median-specific configuration (Figure 3.5), we have the following elements and the respective operations which should be applied:

- the undesirable segment, which will be replaced by its perpendicular through the middle
- the direct neighbours which will be removed
- the second degree neighbours which need to be modified (with either the start or the end as anchor points) by intersecting with the aforementioned perpendicular

Determining the equation of of segments, based on their extremity nodes

The initial step is to determine the equations of the lines (in the slope-intercept version) which pass through the undesirable segment and through the second-degree neighbours.

In general, considering that a segment is formed by two end-points: $P_1(x_1, y_1)$ and $P_2(x_2, y_2)$, the equation of the (infinite) line which passes through those two points is determined as:

$$\text{Slope: } m = \left(\frac{y_2 - y_1}{x_2 - x_1} \right) \quad (3.1)$$

3. Methodology

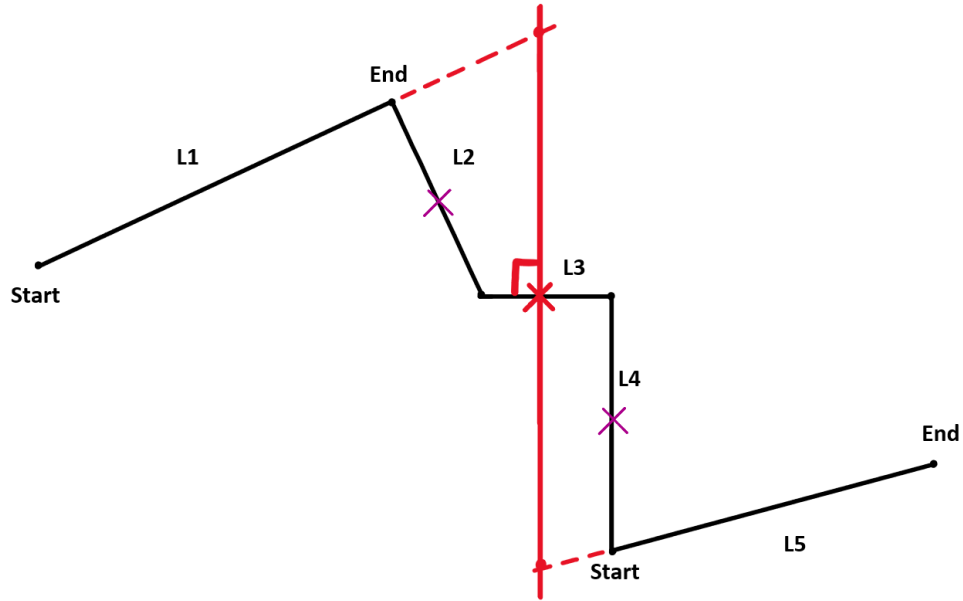


Figure 3.5.: Usual Configuration of a Median Simplification

By plugging in the slope into the general line equation ($y = m * x + b$) and replacing x, y with P_1 or P_2 , we can then determine the y-intercept as:

$$\text{Y-intercept: } \mathbf{b} = y_1 - m * x_1 = y_2 - m * x_1 \quad (3.2)$$

$$\text{Final Line equation: } \Rightarrow y = \mathbf{m} * x + \mathbf{b} \quad (3.3)$$

A bit of an issue may be noticed in these formulas, namely the fact that, in the case of vertical lines, the slope would end up being infinity. For the scope of this graduation paper, the solution to this problem is given in the form of a coding fix, which is explained later on in the [Defining operations and simplifying](#) Section of the next chapter.

Computation of the perpendicular bisector of a segment

Once we have determined the 'actors' and their specific characteristics (such as line equations and anchor points) in the previous subsection, it is then time to start the replacement process. The first step is to determine the perpendicular bisector (i.e. the line which passes perpendicularly through the middle) of the undesirable segment. Considering that the slopes of two perpendicular lines are the negative reciprocals of each-other² we can then determine the value of the new slope as:

$$\text{Slope: } \mathbf{m}_{\text{perpendicular}} = \left(\frac{-1}{m_{\text{original}}} \right) \quad (3.4)$$

²[https://study.com/academy/lesson/perpendicular-slope-definition-examples.html](https://study.com/academy/lesson/perpendicular-slope-definition-examples.html#:~:text=The%20slopes%20of%20two%20perpendicular,%2B%20%20is%20%201%2F2.)
#:~:text=The%20slopes%20of%20two%20perpendicular,%2B%20%20is%20%201%2F2.

3.1. Implementation of the Samsonov-Yakimova orthogonal Line Simplification algorithm in isolation

Once we have the mid-point coordinates (where $P_{\text{mid}} = (\frac{x_{\text{start}}+x_{\text{end}}}{2}, \frac{y_{\text{start}}+y_{\text{end}}}{2})$), we can then plug the slope and the point into the line equation, similarly to Equation 3.2, in order to determine the y-intercept. Thus, the final equation for our perpendicular bisector would be:

$$\text{Perpendicular Bisector equation: } \Rightarrow y_{\text{perp}} = m_{\text{perp}} * x + b_{\text{perp}} \quad (3.5)$$

Finding the intersection between two line-equations

Now that we have this perpendicular equation alongside the equations for the second degree neighbours, it is time to determine the intersection point between each neighbour and the perpendicular line. For this, we basically have to mathematically determine the intersection point between two line equations (Figure 3.6). For this, the following formulas are applied:

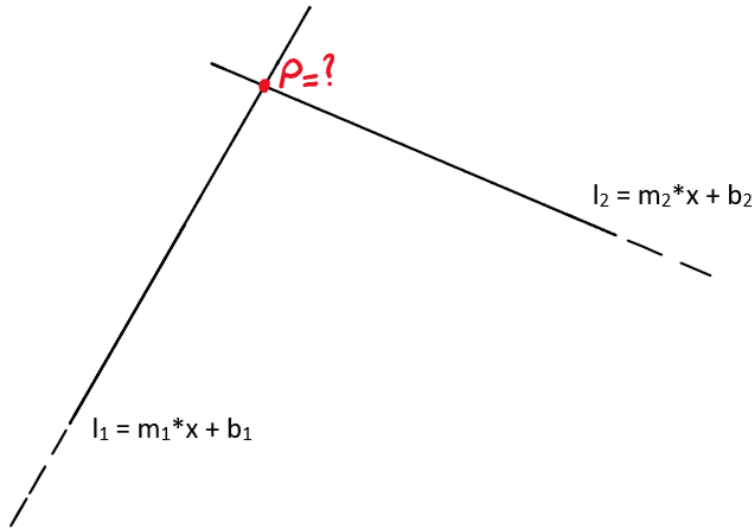


Figure 3.6.: General structure of the intersection between two line equations

Considering that our lines are defined as:

$$l_1 : y = m_1 * x + b_1$$

$$l_2 : y = m_2 * x + b_2$$

We know that, at the point of intersection, the y coordinate is equal, which means that the two equations can be equaled as:

$$m_1 * x + b_1 = m_2 * x + b_2$$

Which can then be rearranged in order to extract the value of the x coordinate:

$$m_1 * x - m_2 * x = b_2 - b_1 \Rightarrow x = \frac{b_2 - b_1}{m_1 - m_2}$$

And now x can be replaced into either l_1 or l_2 in order to determine the y -coordinate.

$$y = m_1 * \left(\frac{b_2 - b_1}{m_1 - m_2} \right) + b_1$$

3. Methodology

Thus giving us the intersection Point:

$$P = \left(\frac{b_2 - b_1}{m_1 - m_2}, m_1 * \left(\frac{b_2 - b_1}{m_1 - m_2} \right) + b_1 \right) \quad (3.6)$$

We apply Equation 3.6 once between the left second-degree neighbour (which has as anchor Point - End) and the perpendicular line - resulting in intersection Point $P1$, and a second time between the right neighbour and the same perpendicular line, with result $P2$. Now, the final step in the process is to modify the structure of the segments by modifying their anchor points with their respective intersections, and creating a new segment from $P1$ (start) to $P2$ (end). The undesirable segment will be removed fully, alongside both of its direct neighbours.

3.1.2. Diagonal Implementation

This operation always occurs in the B and C cases from the different possible simplification scenarios (Figure 3.3). In this situation, we have one segment (always either the first or the last) from which we would like to only keep the anchor point, and a segment which should be modified, whose linear equation will constitute the base on which the perpendicular from the anchor point will be traced.

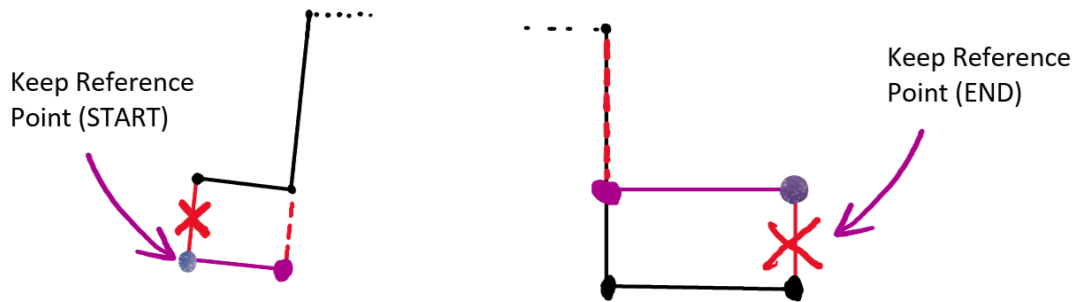


Figure 3.7.: The two situations which may occur in Case B and C, where the "Keep Reference Only" segment (drawn in red with a red cross on it) is removed and a new line is created (with purple) between the anchor point and another segment

Thus, considering that our segment has a slope-intercept form (Equation 3.3), and our anchor point has coordinates (x_A, y_A) , the following step is determining the intersection point on the perpendicular from the anchor (Figure 3.8)

Knowing the slope of the perpendicular line is $\frac{-1}{m}$ (from Equation 3.4), and having one of the points through which this line passes (i.e. the Anchor point), we can then substitute the values in order to determine the y-intercept (same as in Equation 3.2). The intersection point P can be then found by applying the formulas presented in [Finding the intersection between two line-equations](#) between our 'modifiable' line and the just-computed perpendicular linear equation.

Once this has been computed, finalizing the simplification process entails modifying the "Keep Reference" Segment by changing the coordinates opposite of the anchor point (mean that, if our anchor is the start coordinate, we will replace the end location with P), remove

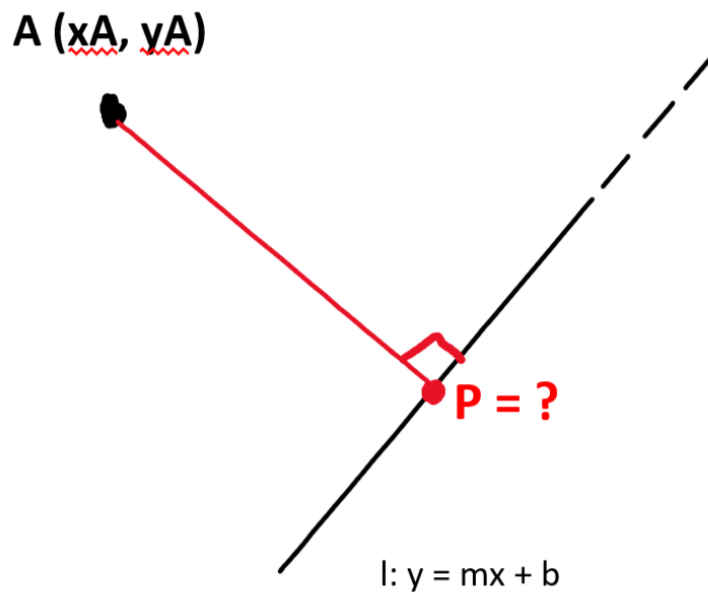


Figure 3.8.: Intersection point on the perpendicular from a point to a line

the undesirable segments and the first degree neighbour in the direction of the segment to be extended, and finally change the anchor point in the 'extend' segment.

3.1.3. Shortcut Implementation

Shortcut simplification occurs exclusively when the number of segments composing a polyline is equal to 4 and the undesirable segment is in the interior (i.e. not a border segment - In this situation, it goes to case C). By far the simplest of the simplification alternatives, it simply requires the removal of the two interior segments (with red crosses on Figure 3.9) from the polyline and connecting the first and last segment directly (from the end-point of the first segment to the starting point of the last segment - with purple on the Figure).

3.1.4. Topological inconsistencies with the median simplification and alternatives

The first issue worth mentioning is the Median Simplification itself. The authors do mention that, although the Median algorithm is generally preferred as it equalizes the propositions between the neighbours, it might lead to topological errors (from a local perspective, meaning self-intersections). Consider the situation in Figure 3.10 (left), where the purple segment is the undesirable one. Due to the strange configuration of this polyline, there the segment to be removed and its neighbour to the right have an incidence angle close to 180° , meaning that the two segments are almost co-linear. As a consequence of this, the line passing through the second degree neighbour to the right (with the Anchor as Start) looks to be almost parallel

3. Methodology

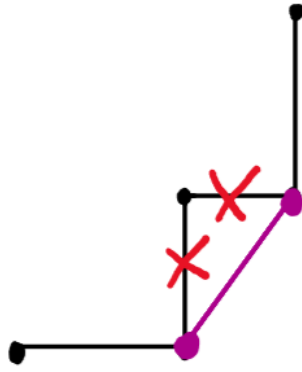


Figure 3.9.: Shortcut Simplification Configuration

with the orthogonal to the undesirable. They do meet-up at some point, however that is so far away that the rest of the segments remaining in the polyline look almost squashed (Figure 3.10 (right)). And while this is not a situation which directly results in a self-intersection, it is certainly quite undesirable.

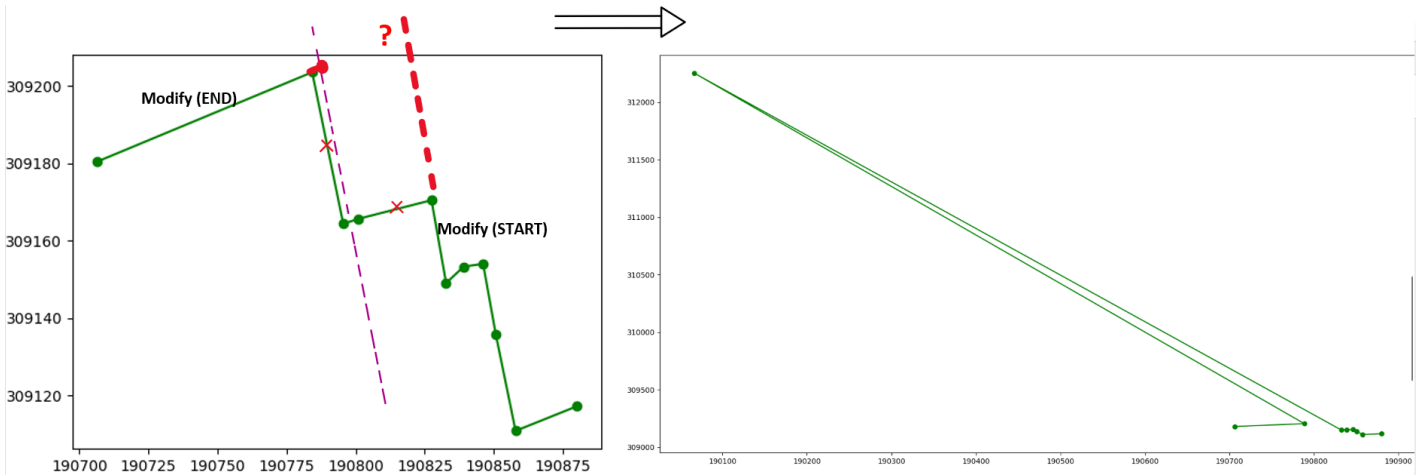


Figure 3.10.: Edge case where the Median Simplification results in a very strange outlier

For this reason, the Shortcut simplification can also be used as an alternative. In order to implement this in our workflow, the following changes are being performed (Figure 3.11):

- decide upon which of the two segments with an “extend” operation will keep its operation, while the other will remain unmodified (drawn in Orange)
- The anchor point of the latter segment will serve as a basis for determining the intersection as presented in the sub-chapter “[Computation of the perpendicular bisector of a segment](#)”

This means that the result of the simplification will be different, depending on the decision performed. At a first glance, it seemed that making the aforementioned changes would have a direct influence on which of the neighbouring faces would gain new territory and while the other would end up losing some of its (as it is the case when we’re working with a Z-like configuration, in Figure 3.12. However, this does not seem to be the case for the segment used

3.1. Implementation of the Samsonov-Yakimova orthogonal Line Simplification algorithm in isolation

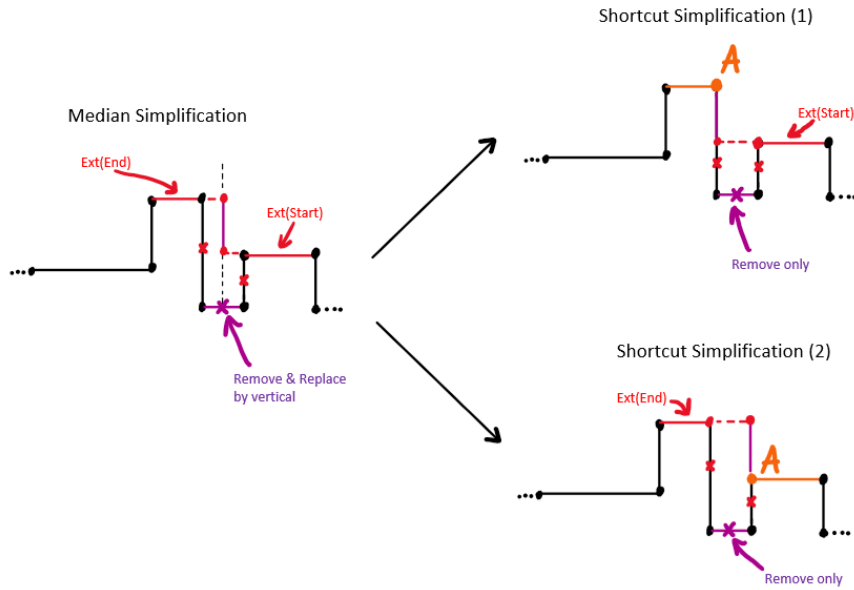


Figure 3.11.: The two alternatives to the Median Simplification

earlier in the explanation (Figure 3.11), where the left face loses some of its area regardless of which segment extension operation is being kept. For this reason, making this decision is no longer a sufficient requirement for determining how the left/right face will change, so extra analysis is required.

While the area which is adjusted (seen hashed with purple in Fig. 3.12) can be determined by the end-nodes of the undesirable segment, the removed Anchor location from the modifiable segment, as well as the newly-determined intersection point (forming a rectangle), this does not give information to which of the two neighbouring faces this modified area goes to. The alternative solution to this problem would be:

- select one of the two faces
- compute its area, and use that as a baseline value
- compute the area after performing the simplification in the two different scenarios, and express that values as a difference $\Delta area_1$ and $\Delta area_2$ between the baseline value and itself
- depending on whether that chosen segment is considered for enlargement or not, decide between the biggest or smallest Δ .

However, if a certain face is set to be enlarged, this does not automatically mean that this will be possible. For example, if we consider the Right case in Figure ??, and say that the face on the left belongs to a class which we might wish to keep visible for longer on the map (for example, a building). In this particular situation however, adding more area to our original face is not possible. The only decision that can be made is a “least of two evils” kind, and going with the bottom version would result in a small decrease of size. This is of course not ideal, but a separate condition could be introduced such that, if both Δ s are negative, then a different operation could perhaps be applied, depending on how important it is for certain classes to always expand or not.

3. Methodology

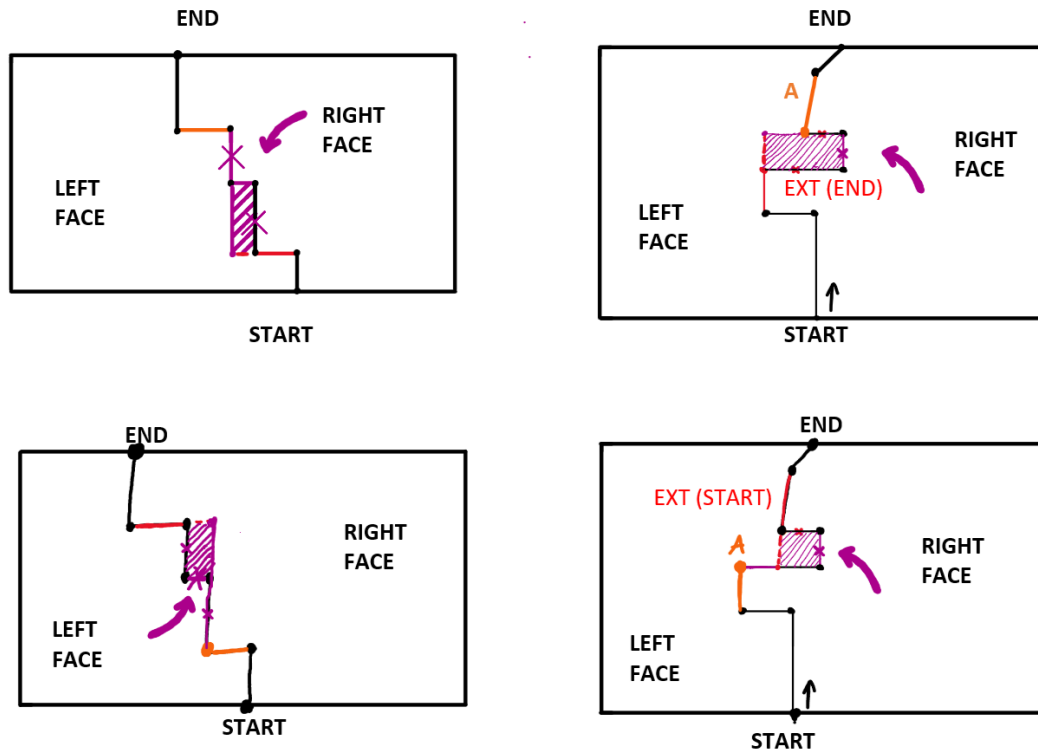


Figure 3.12.: Simplified lines as seen in the bigger context of neighbouring faces. Left: Z-like configuration; Right: the configuration in Figure 3.11

3.2. Integration with the broader tGAP-system and topological aspects

The next step in the methodology, after making sure that the Orthogonal line simplification fully works in isolation, is to integrate it in the broader workflow of the tGAP generation. The result of this phase would be that the [Samsonov and Yakimova \[2017\]](#) Algorithm can replace the previously-used Reumann-Witkam Simplification as the solution used for simplifying polylines after the other split and merge operations. However, this is only an intermediate step, and it is not our desire to completely replace RW with the new SY, but to integrate both of them: further information regarding the particularities of choosing between two or more algorithms is discussed in the next section. Regardless of this, it should be noted that this discussion is valid on both the tgap with SY, as well as tGAP working with both SY and RW.

3.2.1. Correctly detecting intersections with external elements

While there are a number of issues and points which should be kept into consideration when introducing a new edge-simplification algorithm into the tgap-generation workflow, the main thing we wish to avoid are the topological inconsistencies. When we take a line, in isolation,

and modify it such that it becomes simpler, when introducing it back into the planar partition, the main issue that exists would be that our changed geometry overlaps or intersects other neighbouring elements. For this reason, further checks are required.

When working with the RW Simplification, the solution would have been as follows: when choosing a particular triangle (with the smallest area) to collapse, it would have been a sufficient condition to simply check for any points which may lie inside the collapsible triangle. The points could be taken from the QuadTree Structure, and what is left is simply a check to see if there are any points from the QuadTree inside the area of the triangle (Figure 3.13 - Left, where our modification is determined by the dashed-red line, which flattens the triangle, and where the neighbouring element is drawn in purple). However, this is no longer a sufficient condition when it comes to SY simplification.

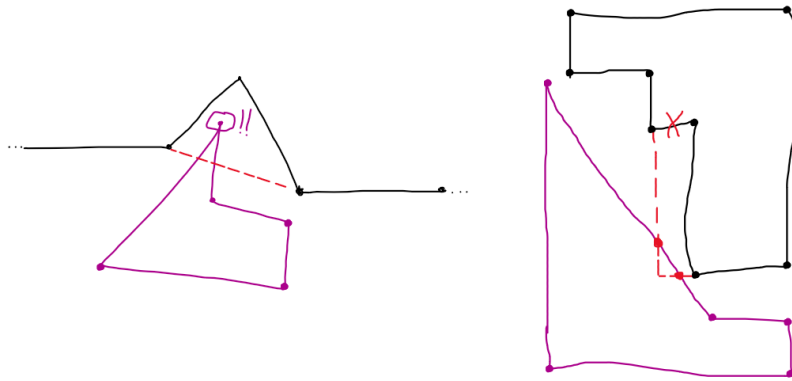


Figure 3.13.: Checking for intersections in the Planar Partition by looking for the points in QuadTree in the situation of RW (Left - Guaranteed to succeed) and SY (Right - Can fail sometimes)

Considering the situation in Figure 3.13 - Right, we can see that the newly-created area (determined by the dashed-red line) ends up crossing the neighbouring element (drawn in purple) in two different points. However, there are NO edges inside the changed area. Due to this issue, we need to come up with another solution.

Another solution which can be considered would be the following: when simplifying a certain edge, check the resulting geometry against all of the constituent edges which form the left and right faces (with the obvious exception of itself). This way, we can guarantee that any other neighbouring edge does not intersect with our own. This is the solution which is implemented in the methodology.

This can be seen in Figure 3.14 how this operation is being performed. Basically, the algorithm ends up detecting two locations where our simplification intersects one of its neighbouring edges (depicted in the figure with purple crosses), which in turn rings the alarm bell that an alternative should be found (or the operation should be performed after other changes have been done). There are, however, a number of issues with this alternative as well, the first one being the fact that it (can) involve a very large number of checks. For example, when we consider the situation where our edge to be simplified lies on the boundary with the world-face (i.e. the exterior of our planar partition): do we end up checking ALL of the elements which constitute it, thus ending up having to do an incredibly large amount of intersections,

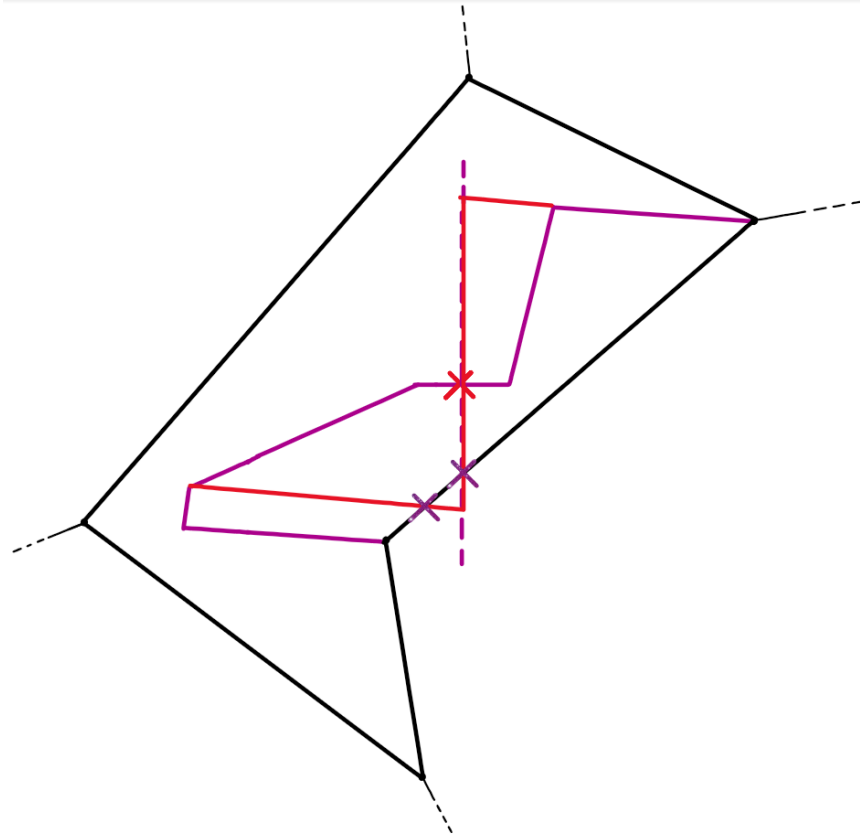


Figure 3.14.: Example of how the intersection with first degree neighbours is being performed, where the geometry to be simplified is shown in purple, and neighbouring edges with black.

or do we consider it as an exception, and don't check it at all? In my research, I found that the best middle-ground when dealing with an element at the edge of the planar partition is to just check those edges which also border have the world-face, while at the same time ending in a common Node. This, however, still does not fully guarantee success , while at the same time still keeping the complexity quite high. As this module is the one which has been implemented, further details on its complexity are given in the [Results and analysis](#) Chapter.

On the other hand, there is another issue which can occur when applying this method, albeit quite rare, and occurring solely (under the tests performed throughout this thesis) when applying only the SY simplification for all elements, regardless of their type, is that the change which we apply, under SY simplification, to a particular edge can be so "drastic" (in situations similar to Figure 3.10 - even though the resulting shape is simple when it comes to the number of constituent nodes that it has, it is certainly not simple when it comes to its overall geometry), that it may end up going over the first degree neighbours (the faces to the left and right), directly onto the territory of the second degree neighbour (such as the left neighbouring face of an edge which is part of our left neighbouring face). This might be solve by checking both the second (as well as the third?) degree neighbours, but at this point in time,

3.3. Integrating multiple algorithms in the tGAP generation

we are introducing a lot of complexity in our solution, which is far from desirable.

The best solution for solving this issue would be to simply implement a geographic structure, such as an R-tree which would store the position of all our edges, and which could easily be used when it comes to checking the neighbours. Further information on this is given in the [Conclusions and Future Work](#) Chapter.

3.2.2. Improving the selection of the edges to be simplified

Deciding which edges should be simplified, based on a particular criteria and at a particular step, is another important feature of the tGAP generation algorithm. This module uses a number of different principles to classify the edges and determine whether or not they are still suitable for visualization, while keeping in mind the medium through which we will visualise the result: a computer screen.

In the original version of the algorithm, the solution works as follows: for all the edges which exist in our planar partition (which are formed by at least two segments), we take all groups of 3 consequent nodes. For each grouping, we compute the height based on the area formed by all three points, divided by the half of the base (as determined by the distance between the first and last nodes in our sequence). The smallest value is saved from each edge, and that value will also represent the priority value when inserting that particular edge in a Priority Queue. Afterwards, all the edges, alongside their priority, are checked, at each step of the process, against a simplification threshold, which is based on the resolution of the screen, in accordance to the scale at which the process is at.

This method makes a lot of sense, due to the fact that, when considering a particular edge on a screen, when zoomed out, it is quite difficult to distinguish very short triangles (with a very small height from the base to the top), those looking almost as if they were continuous. At the same time, due to the nature of how the Reumann-Witkam simplification works (Chapter 2), those very short triangles will also be the ones which are removed. This, however, poses a significant issue when introducing another algorithm such as Samsonov-Yakimova into equation: the process through which we select a suitable edge to be simplified may not be equivalent with the shortest segment that we decide to remove from that edge. Thus, a simplified edge may end up being sent back with the exact same selection value as before, as the part with the shortest triangle may not have been simplified.

A solution to this comes in the form of adapting the priority value algorithm. In the case of Samsonov-Yakimova, a good solution can be the selection of the value of the shortest segment in a particular edge, and comparing this against the resolution (which is still a value based on the scale, as it was in the previous variant of the algorithm). It does make sense from the point of view of the user, as, when looking at a map with a lot of zoom, very short segments may not even be well displayed in a pixel, so removing them first should indeed be a priority.

3.3. Integrating multiple algorithms in the tGAP generation

The final step in the process in adapting the algorithm in such a way that, when having more than just one simplification solution implemented, it can decide which is the most suitable option from the bunch. The first solution, and also the one which is implemented in this graduation thesis, is a "semantic-based selection of line simplification alternatives": it uses the

3. Methodology

Border between	Cls_1	Cls_2	...	Cls_n
Cls_1*	Cls_1_cA	$f(1_cA, 2_cA)$...	$f(1_cA, n_cA)$
Cls_2*	$f(2_cA, 1_cA)$	Cls_2_cA	...	$f(2_cA, n_cA)$
...
Cls_n*	$f(n_cA, 1_cA)$	$f(n_cA, 2_cA)$...	Cls_n_cA

Table 3.1.: Compatibility matrix of multiple classes

feature classes of a particular dataset (for example: what topographic type a particular face is) to decide which is the most suitable line simplification which could be implemented. It should be noted that, in order to use this variant, it is a prerequisite for a particular dataset to be enriched with such information - i.e. the dataset should be attached to a proper topographic map [Kent \[2009\]](#). Using an unclassified dataset will not work with this method.

Considering we have a number of n distinct classification categories (for example roads, rivers, agricultural lands, etc.), the first thing to do is to assign a specific simplification algorithm for each of the aforementioned feature classes, in accordance to which solution is more suitable for which particular case (where multiple classes can have the same algorithm). In the particular example of our methodology, we consider buildings to be simplified using the Samsonov-Yakimova method, and all other cases to be dealt with using the Reumann-Witkam solution.

Afterwards, there are a number of variants which could be implemented: the first, and simplest, would be to create a priority list, where all the available feature classes are ranked in accordance to their importance ($priority_queue = [class_a, class_b, ..., class_n]$). Using this, when having an edge which neighbours two distinct faces, the more important one will dictate which algorithm will be selected in the simplification of that particular edge. The problem, however, with this solution, is that it may be a bit too simple and too straight-forward. One simple improvement of it can be using a randomizer with a particular predefined chance (for example, 75%-25% in advantage for the more important class), but this might result in a result which is not desired. This being said, the fact that this solution is not complex is not an issue in itself. The issue arises from the fact that the algorithm becomes very rigid, and due to this can end up choosing the wrong simplification, and can thus result in an inaccurate map, from the perspective of the user.

Another, more complex variant of the solution, is in the form of a compatibility matrix, which is based on a (manual or automatic) criterion. Consider the Table 3.1, where the right face of an edge is represented by the rows, while the left face by the columns. Cls_number_cA (or shortend as $number_cA$ in functions) represents the "chosen Algorithm" for that specific class.

While in theory a good way of deciding which is the best way to go about a simplification, the function itself is not as simple to determine. One can request a particular user to fill this table manually (for example, deciding in each case which one takes precedence), in accordance to their needs.

4. Aspects related to Implementation

After introducing the general concepts behind how the methodology works, in this chapter it is time to take a look at the particularities behind the implementation of the methodology. Firstly, the software solutions which are used in the graduation project are introduced, followed by a brief description of the datasets which will be ingested by the final workflow. Afterwards, a more in-depth discussion into the particularities of the Samsonov-Yakimova Implementation is had, followed by a chapter on the process of integrating everything into the tGAP Workflow. In the final section of this chapter a brief introduction into the analysis software is presented, which will be then used as a basis for the next chapter.

4.1. General Software details

The main chunk of the work done throughout this graduation thesis has been performed as an extension to the already-existing tGAP generation software¹, using Python² as the preferred programming language, as it can offer a number of versatile tools for geographic-based software development. The final version of the code, created for testing and validating the concepts presented in the previous chapter, exists as a fork of the main tGAP software, and it is hosted on my personal GitHub account³. At the same time, the analysis solution, which is to be implemented separately from the main tGAP Generation algorithm (and which will be used to get the graphs in the [Results and analysis](#) Chapter) is also hosted on my GitHub account, but in another repository⁴.

One important Python package worth mentioning, which is extensively used throughout the development process of this solution, is Shapely⁵, which is used for analysis and manipulation of various geographic data (such as planar features). Shapely has been developed using the GEOS⁶ library.

All of the data used in the project has been stored in a PostgreSQL Relational Database Management System⁷, in a Database which has been enriched with the PostGIS extension, due to its spatial characteristics, including the compliance with the Open Geospatial Consortium's "Simple Feature"⁸.

Lastly, the QGIS Software package⁹ has been used for visualisation of the resulting solution and its visual analysis. As reference for the discussion to be had throughout this chapter, the

¹<https://github.com/bmmeijers/tgap-ng>

²<https://www.python.org/>

³<https://github.com/erbasualex/tgap-ng>

⁴<https://github.com/erbasualex/tgap-analysis>

⁵<https://shapely.readthedocs.io/en/stable/manual.html>

⁶<https://libgeos.org/>

⁷<https://www.postgresql.org/>

⁸<https://postgis.net/features/>

⁹<https://qgis.org/en/site/>

4. Aspects related to Implementation

functioning principle of the tGAP generation software is depicted, as an activity diagram, in Figure B.2 in Annex.

4.2. Datasets

The overall structure of this type of data is represented, inside a geo-spatial database, as two separate collections of elements: the first one depicts all the individual regions or areas (henceforth referred to as 'faces') while the second collection keeps record of all the borders between the said faces (also known as 'edges'). In order for the algorithm to function as intended, the dataset used needs to adhere to a couple of principles. Otherwise, it would not be possible to apply the generalization process. The most relevant requirements that any data should follow such that it can be successfully plugged into the algorithm are as follows:

- All edges should be simple (i.e. no self-intersections present), and no two edges may intersect each-other
- No two faces shall overlap, and there should not be any gaps in the datasets (i.e. regions which are not recorded as a face)
- All faces need to be enriched with a classification category, giving it a more semantic characteristic (meaning we no longer work with just purely-geometrical objects, but instead with cadaster-relevant places in a particular area)

For this graduation process, a subset of the TopNL dataset has been used, covering an area of approximately 60 square kilometers, in the south of the Limburg Province, in the Netherlands (Figure 4.1). This dataset is part of the Topographic Base Registration service ('Basisregistratie Topografie' in Dutch, or BRT) which is collected, administered and provided for public usage free of charge by the Dutch Kadaster Agency¹⁰.

Out of all the publicly-available collections of geographic data provided by the Dutch state agency, the Top10NL¹¹ has the highest level of detail, and it is designed to function at a scale from 1:5,000, up to and including 1:25,000. At the same time, it is classified with a number of useful semantic attributes, such as roads, railways, building, agriculture and so on, making it perfect for the scope of the algorithm which is introduced in this paper.

The dataset, which encompasses an area of approximately 75 square kilometers, is then further more divided into smaller datasets, as seen in Figure 4.2. The sub-division is performed with a factor of three, such that each subset (with the exception of Small Test, which was used exclusively for making sure the software can run during the development process) has three times less the number of faces as its bigger counterpart.

Further detailing how the dataset is being stored in the database, the following particularities are relevant to point out:

– The edge table has 'ogc_fid' as the Primary Key, which represents the entry of a geometry in the database, and another 'edge_id' which is used for differentiating the various edges in the planar partition. The geometry is defined as a LineString, in the Dutch Coordinate Reference System¹², and that has a spatial index on it¹³. It is defined by a start and end node, and a

¹⁰<https://www.kadaster.nl/-/brt-catalogus-productspecificaties>

¹¹<https://www.kadaster.nl/zakelijk/producten/geo-informatie/topnl>

¹²<https://epsg.io/28992>

¹³<http://postgis.net/workshops/postgis-intro/indexing.html>

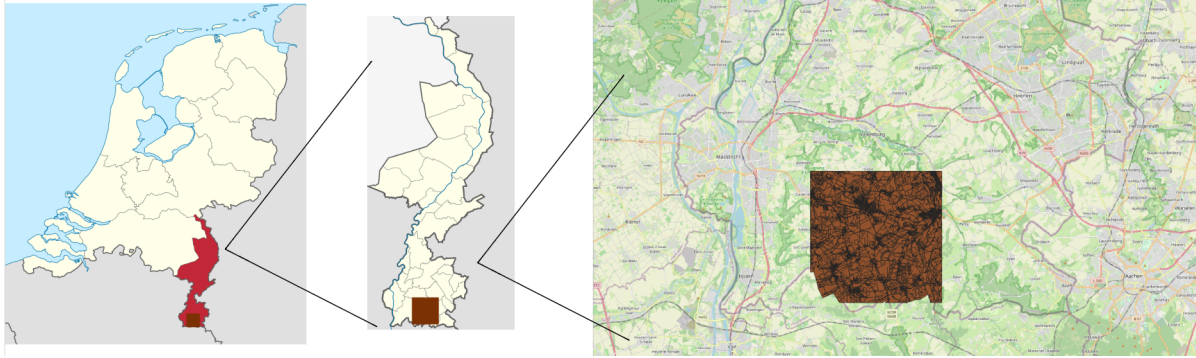


Figure 4.1.: The area used as dataset, displayed on the national (left), regional (center) and local (right)-level

left/right face - all saved as ids.

– The face table has a similar Primary Key, but two different kinds of ‘geometry’, a bounding box (saved as a Polygon) and a center node (stored as a Point), both of which also having a spatial index on it.

Lastly, some initial statistical information regarding the datasets can be found in Table B.1, in the Appendices. Here, it is easy to see the factor of 3 when it comes to increasing the size of each dataset (when it comes to the number of buildings). Another thing worth pointing out is the fact that, regardless of the datasets, initially, there are always more points than edges at a factor of about 5-6%.

4.2.1. Main Data Structures used in the General tGAP generation Algorithm

The design behind the data structures which are used in the overall generalization algorithm was conceptualized in the original version of the tGAP Generation software, and was not a direct part of this dissertation thesis. However, briefly explaining them is necessary, as the implementation itself is based on this stem.

Firstly, one needs to keep in mind the fact that, in a generalization process, the data undergoes frequent modifications of the way the area and line objects are represented. For this reason, instead of storing the area objects as simple feature geometries, we make the decision to store the composing primitives of the areas (i.e. their boundaries), and define the faces (another term for area) as a collection of boundaries/edges. These segments are not allowed to self-intersect or intersect with each other, except in their end-nodes.

The edges are defined using a start and an end node, and the geometry in-between these end-nings. In the end, the collection of these non-intersecting boundaries forms a complete planar partition. Overall, the dimensional primitives, henceforth defined as the “*wheel topology*”, is constructed in the code as follows:

```
Node = namedtuple(
    "Node",
```

4. Aspects related to Implementation

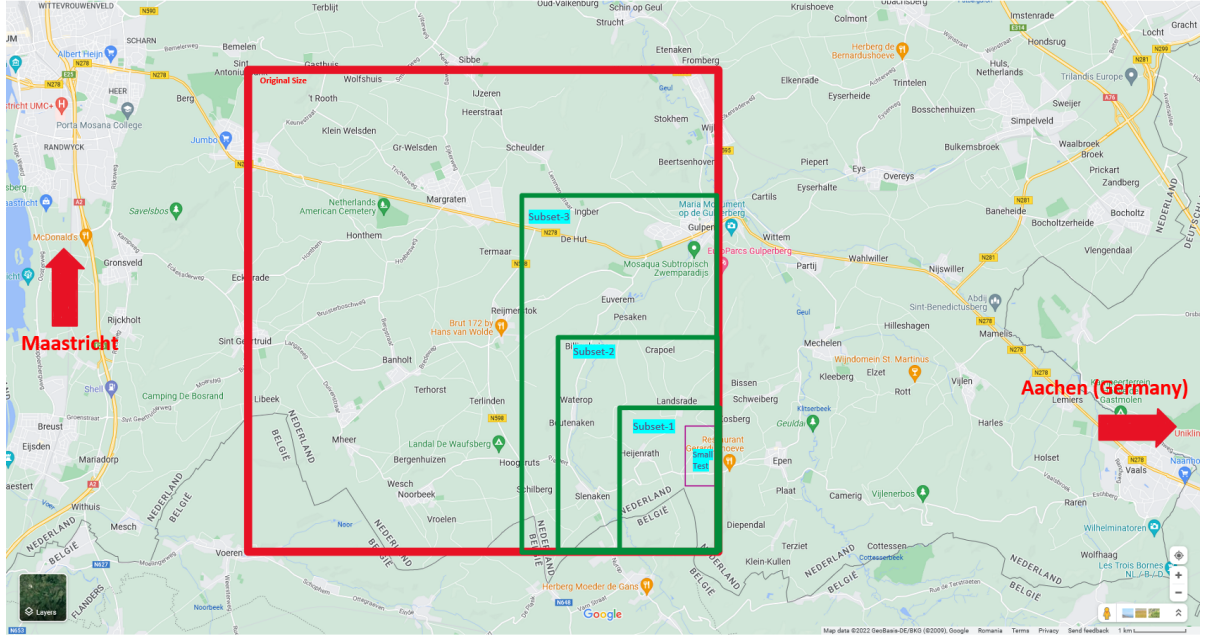


Figure 4.2.: Sub-divisions of the original dataset

```

    "id, geometry, signed_edge_ids"
)

Edge = namedtuple(
    "Edge",
    "id, start_node_id, start_angle, end_node_id, end_angle, left_face_id,
    right_face_id, geometry (polyline), info"
)

Face = namedtuple(
    "Face",
    "id, mbr_geometry (rectangle), pip_geometry (point - center of surface),
    signed_edge_ids, info"
)

```

Where `mbr_geometry` stands for a tightly fitting box around all the edge geometries of the face, and `pip_geometry` represents a point located in the interior. All the 'signed' identifiers which can be found in the structure above are used to represent direction. By taking its' two's complement, we get the opposite direction of the same. These signs are used when traversing the structure, as flipping the primitive can help to easily find the opposite direction. In the case of node, the sign represents all the Edges which are either going in our out. For the case of faces. the edges define its boundary, and sometimes may need to be swapped using the two's complement.

4.3. Line Simplification Implementation

The Samsonov-Yakimiova line simplification algorithm has been developed as an independent module, which can be plugged in and which can replace any already-existing simplification component, or can be simply used in isolation with any edge which is in our desired format. It has been developed in multiple components, such that it adheres to modularity programming principles [Macdonald, 2020], including a unit which coordinates the simplification and handles exceptions which may occur (taking the role of a handler), the SY-specific data structures which saves the original structure, and (tries) to convert it, as well as a couple of helper classes created to aid with utility functions and various constants.

The general class diagram can be found in the Annex in Figure B.8. There are four main components in the system, the main one being the handler, which coordinates the entire process. The ‘data structures’ module contains the logic behind how the geometry is temporarily stored, with the purpose of being manipulated, while the ‘Constants’ module defines all of the operations and properties which are used in the simplification process. Lastly, the ‘geometry’ module implements a number of functions which are used in the geometrical operations applied in the generalization process, as well as a data structure designed for handling the line-equation, and all of its exceptions (based on our used definition for a line).

4.3.1. Coordinating the simplification

The first piece of the puzzle coordinates the simplification process by the simplification starts by taking an Edge object, and retrieving its start and end points. These nodes will be later used in the intersection check. The next step in the process is to convert the geometry into Shapely geometry, as well as converting all constituent points to Shapely Points.

In comparison to Reumann-Witkam, where we were analysing sequences of (three consequent) points, we now need to look at the problem from the perspective of segment lengths: for this reason, a Segment Collection structure is being created, which keeps a record of all of the constituent pieces of a complex edge. Further details on this structure are presented in the following section. For now, it is worth mentioning that our handler class creates such a structure from the original Edge.

Using this segment collection structure, the handler now tries to create its simplified version, while at the same time keeping track and handling any exceptions which may arise, including issues which can be caused by the generation of the Segment Collection itself. The simplified version of the edge, which is still in its Shapely form, can then be checked for any self-intersection, using the ‘is_simple’ property of Line Strings [Gillies, 2022]. This last step ensures that the result that we got is correct from an individual point of view (i.e. not taking into account all other elements of the planar partition).

Now, it is time to take a look at the individual processes that occur in the Segment Collection creation, and how the simplification itself is performed.

4.3.2. Used Data Structures and pre-simplification classification

As mentioned in the previous section, working with groups of three consecutive nodes is no longer the way to go when employing the Samsonov-Yakimova simplification. For this

4. Aspects related to Implementation

reason, a data structure which takes the segment as a basic unitary element needed to be defined.

We define a (Simple) segment as an object which is initialized by its constituent nodes, and which has a certain length. An advanced segment is an improvement to the aforementioned simple segment, to which a specific operation is attached. We define an operation for a segment a specific action that will be performed on that particular segment, as follows:

Keep The segment will be preserved in its entirety, and no change will be performed on it

Remove This is the segment that will be removed completely from the simplification perspective, due to it being the shortest one in the edge

Ignore This is the property of the segments neighbouring the shortest one, which will end up being ignored in the final solution. Due to the way the algorithm functions, they will end up being ignored in the final resulting simplified edge.

Extend* A segment with this property will end up being modified, but only from one of its ends - it will result in a segment with the same line equation as before, and one of its ends in the same location, but being larger or smaller according to the change of the other end node

Keep with Reference Point* Similar to the Keep Operation, except we denote one of the ends of this segment as relevant for later operations

Keep only Reference/Anchor Point* Operation introduced specifically to replace the Median simplification, will only retain one of its relevant points (the anchor), and will replace the original segment with a new one, having a different line equation

Short Interior* Operation used only in the situation of a short edge (where the Diagonal Simplification is being applied), where our edge contains exactly four segments

The operations demoted with an asterisk (*) have an extra property of "reference point": this denotes one of the ends of the specific segment (either its start or its end) as relevant for a specific action.

We define a "Segment Collection" as a list of Advanced segments, which also has attached a Transnational Manager, and which can perform the operation of simplify and simultaneously keep track of any changes which may occur, and reverse them should the simplification fail under any circumstances. All these operations under Segment Collection are coordinated by the handler mentioned in the previous section.

The last relevant structure worth mentioning is the Line Equation, which converts a segment from our original shape (defined by its end-points) into a y-intercept equation, used for computing various operations.

Using the previously defined operations, the next step in the process is to classify a segment collection in accordance to the positioning of the shortest segment (which is the one which will be removed in the process). The main workflow of the process is explained in the Algorithm 1.

We consider an interior segment as one which is neither the first one nor the last in the Segment Collection. We define these to be "border segments". In cases where the number of segments in our Edge is strictly greater than four, we also define the neighbours of the border segments (i.e. the second or the second to last segment respectively) as being "next-to-border" segments. We also define a circular edge as one which starts and ends in the same node.

4.3. Line Simplification Implementation

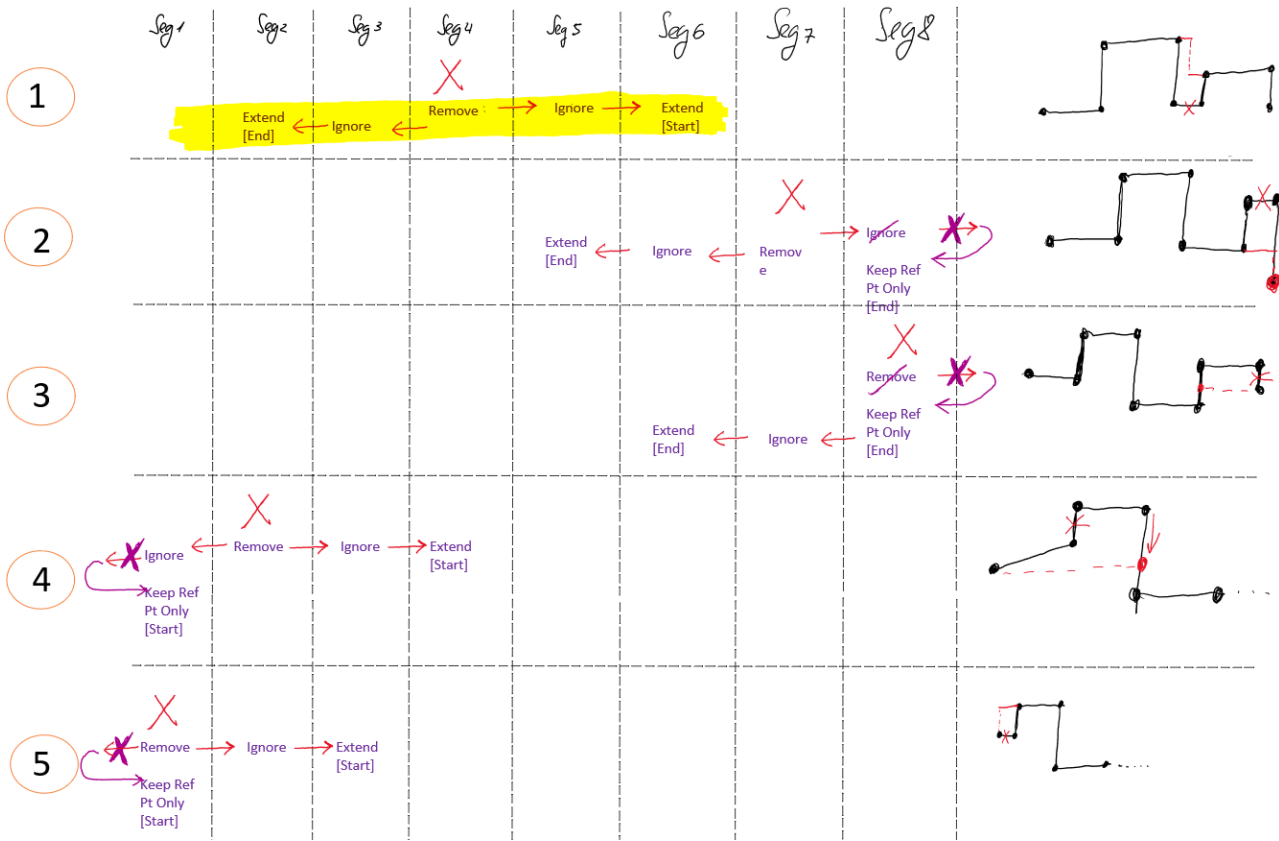


Figure 4.3.: Categorising Cases for an Edge with more than four segments

Although the algorithm itself seems a bit strange, its functioning principle, in a nutshell, is as follows: first, it does some initial checks, to see if operations can be attached on that particular edge (we cannot apply the Samsonov-Yakimova algorithm on edges with less than four segments, or on circular edges with less than 6 segments). Afterwards, it works on detecting a few cases, the first one being the Diagonal simplification, which works only on edges with exactly four segments. Should that not be the case, it then starts the process of analysing the shortest segment alongside its first and second degree neighbours to its left and to its right. A good example of how the algorithm works in these cases can be found in Figure 4.3. Case 1 shows the normal case. Problematic cases occur when the segment to be removed is positioned either at the end of the edge (Case 2 and 3), or at its start (Case 4 and 5), situation where a number development decisions had to be taken: here, the Shortcut Simplification will be applied, by selecting the end-Node as an anchor (such like the Endpoint case from the theory of SY).

4.3.3. Defining operations and simplifying


After the classification has been performed (and no exceptions or issues have been returned from it), the next step in the process is to perform the simplification itself. The main operations which have been attached to each segment will function in accordance to Figure 4.4.

4. Aspects related to Implementation

- Ignore -> don't do anything to it

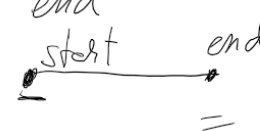
- Extend Segment Start → 

- Extend Segment End → 

- Replace Segment but Keep Starting Point 

- Replace Segment but Keep End Point 

- Keep Segment but Consider Start as Reference Point



- Keep Segment but Consider End as Ref. Point

- Remove completely



Figure 4.4.: The effect of the defined operations on a particular segment

In the original version of the algorithm, all the substitutions mentioned in the Samsonov-Yakimova paper were implemented (Median, Diagonal and Shortcut). However, due to the issues explained in the Section [Topological inconsistencies with the median simplification and alternatives](#) of the Methodology, the final version of the algorithm contains a substitution module, which, in a nutshell, detects the cases where we would have had two Extend operations (one with a Start reference point, the other with an End one). Following the replacement principle explained in Figure 3.11, we transform the Median substitution to a Shortcut one, based on a Left (or) Right bias parameter, as in the algorithm below. Here, the special operation 'Keep WITH Anchor Point' is introduced for the first time.

The pseudocode for such a function can be found in Annex 2. As you can see there, we introduced two new variables in discussion: the pointID and the lineID. these new elements are basically the ones which end up replacing the old operation, where the lineID will indicate the segment which will end up being extended, while the pointID will be equivalent to the 'Keep with Anchor' operation.

Another point worth mentioning is how vertical and horizontal lines are handled. Due to the version of line equation which has been chosen to be encoded in the solution, namely the y-intercept version, poses a big problem especially when it comes to vertical lines. This is because, in this particular situation, the slope of the line is infinity. This, of course, means that we cannot simply use this value in our computations (as we need the inverse of the slope,

that would mean trying to compute ‘one over infinity’). The best solution would have been to fully refactor the entire software such that it would use the ‘general form’.

4.4. Details referring to the tgap integration

The first point to be discussed in this section is the way the implemented solution makes the decision which algorithm to use. This is one of the most important scopes of the entire projects, and getting this module right results in obtaining a map for the end-user to enjoy.

The first thing to define, as part of this module, is what classes of faces are assigned to which particular algorithm. As we have a limited number of algorithms to a plentiful number of classes, a good option for implementing this would be by using a dictionary¹⁴, with all line simplification algorithms classified in an Enum¹⁵ (or IntEnum). Using the same Enum classification, we can define a priority queue, where the user can give as input which types of locations are more important to them, from the perspective of their particular application. Lastly, after these things have been defined, when it comes time to decide which solution to use for a particular edge, we look at the neighbour list (left and right neighbour). According to our own priority, the category with the highest priority wins and gets to apply its preferred algorithm.

Another point which should be mentioned at this step is how different algorithms handle each other’s data-structures. What I mean with this point is the fact that, in some line generalization solutions, it is needed to keep track of some spatial data using a particular, auxiliary, data-structure (for example, Reumann-Witkam uses the QuadTree storage solution to keep track of the vertices which exist in its surroundings). For this reason, any other generalization algorithm which could be used in its place would need to also update this structure. Otherwise, it might cause functioning issues to the original algorithm. The solution for this is to pass the variable as a keyword argument¹⁶ in all other simplifications. At the same time, after passing this, my solution was to create a Transactional Manager, which keeps track of possible changes in the particular data-structure (QuadTree, in our case), and should the simplification with the other algorithm succeed, then the recorded operations in the transaction are committed. Otherwise, no change is done.

A challenging part when it comes to the integration of multiple line generalization algorithm lies in the way that the different data is structured for the different situations. As mentioned in the [General Software details](#) Section, the Shapely package was used for implementing the segment-based data structure. At the same time, the geometry structure used in the tGAP generation workflow is defined by the Simple Geometry Library¹⁷. For this reason, a number of conversion methods had to be developed, so that interchanging between the two data structures can be possible. These methods are defined in the ‘Geometry’ component of Figure B.8.

As discussed in Section 3.2.1, it is not enough to simply check if any points in the quad-tree are contained in the area which is being modified during the simplification operation. Due to this reason, the solution worked was to, when it comes to checking to see if our resulting simplification is correct, for the case of Samsonov-Yakimova, is to extract all of the edges which make

¹⁴<https://docs.python.org/3/tutorial/datastructures.html>

¹⁵<https://docs.python.org/3/library/enum.html>

¹⁶<https://treyhunner.com/2018/04/keyword-arguments-in-python/>

¹⁷<https://github.com/bmmeijers/simplegeom>

4. Aspects related to Implementation

up the left and right face, and to check, against each of the geometry to see if it intersects with our simplified geometry. This approach may however contain certain short-comings, which are discussed in the final part of this thesis, alongside some better alternatives.

5. Results and analysis

After outlaying the framework in the [Methodology](#) Chapter and discussing the various development components in the [Aspects related to Implementation](#) Chapter, it is now time to take a look some results. We will start by presenting the reasoning behind why it is important to introduce a generalization algorithm into the workflow, followed by a comparison between how the elements look like on the resulting map, when considered individually, from the perspective of the end-user. The last part of this chapter will go over a number of difficulties which may be encountered when working simultaneously with multiple generalization algorithms, and their influence on the efficiency of the workflow.

The datasets which were introduced in the Section 4.2 of the previous chapter will be the ones used for both the visual, as well as the statistical analysis. For the latter, it is worth reminding that each subsequent dataset is larger with a factor of approximately 3x than the one before it. Thus, the results that will eventually be compared, contrasted and analysed should be seen from the perspective of the input.

5.1. The effect of embedding a line generalization algorithm into the tGAP generation workflow

When working on developing a particular map generalization system, the decisions taken by the system's architect in regards to the algorithms which are to be implemented should be considered carefully. Before introducing a particular operation into the workflow, it is important examine the benefits (or perhaps the disadvantages, should that be the case) that it may bring. At the same time, as any extra procedure has an effect of the functionality of the overall workflow, it is good to decide whether or not the changes that the operation brings to the generalization algorithm are worth when taking into account the extra hindrance.

As the result of the generalization operation is to be distributed (and utilized) on the internet, a good question to ask before introducing any line simplification algorithms into the mix would be: "What effect does this decision have on the size of the dataset at each of the steps in the tGAP generation process?". The reasoning behind this question is that, as the transmission of data needs to be as efficient as possible, the ration between the number of objects which are being displayed at a particular scale and the area of the map clip which is displayed on the screen of the user should remain more or less constant (i.e. as the size of the area displayed to the user gets larger, thus showing more and more of the Earth's surface, the total number of elements on the map should theoretically decrease, so that the average number of objects per square meter is the same as before). Too much data at a small scale might cause a bottleneck, as a large amount of objects which is need to be rendered can significantly slow the entire process down.

By looking at the over number of points at each of the steps in the process, we can see how effective each variant of the algorithm is, while also keeping in mind the requirements related

5. Results and analysis

to transmitting the data over the internet. Looking at Figure 5.1, the difference between the two situations is quite clear.

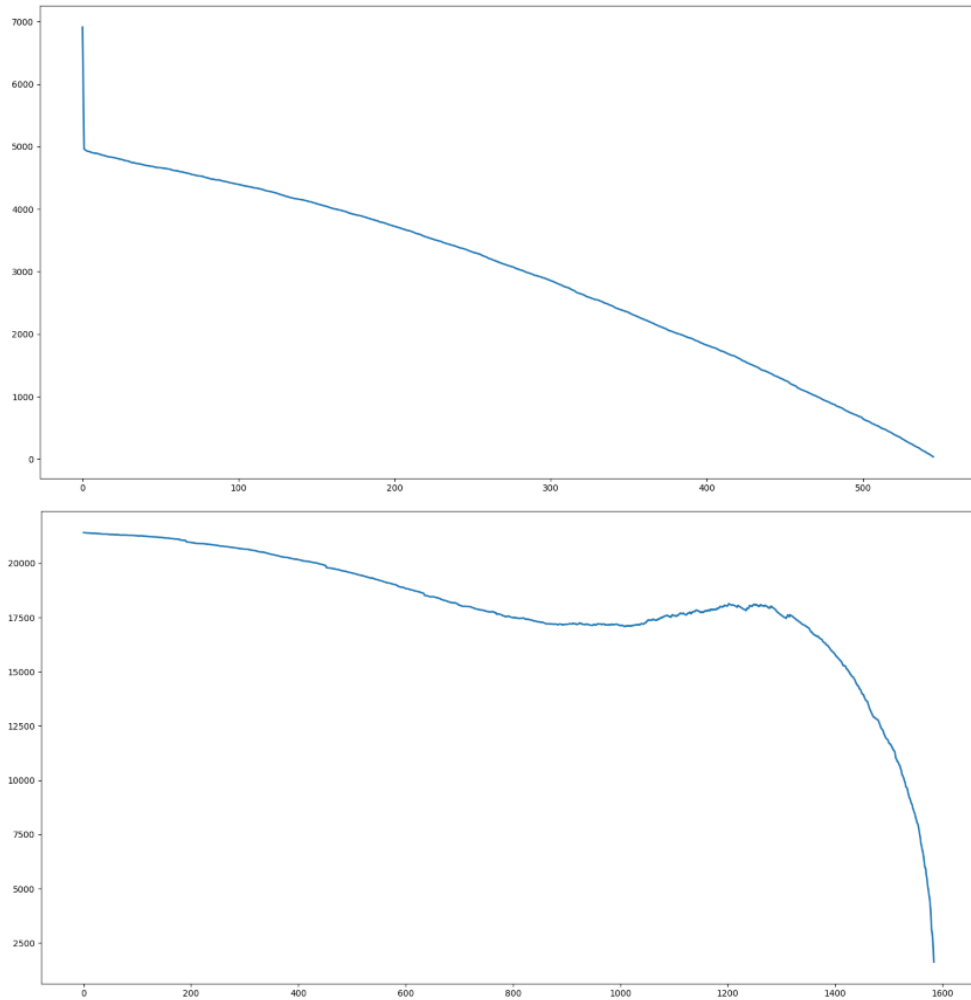


Figure 5.1.: Number of points per step when running a simplification algorithm - Top: with simplification turned on, Bottom: Simplification turned off

When looking at the two cases, we can see the difference: when utilising a simplification in our algorithm, the number of points have a consistent drop. This occurs with the exception of the first step, where there is a huge downfall. This is a clear sign that the initial dataset is way too detailed for our resolution, which in term requires a lot of operations so that the datasets end up at the correct complexity. However, after that initial drop, a smooth decrease in the number of points continued almost linearly until the last step. This structure is ideal in our general workflow, as we end up transmitting a constant amount of data per the size of the zoom level.

On the other hand, looking at the figure when line simplification is turned off, the situation is quite different. The number of points ends up hovering constantly in the upper range, until almost at the end, when there is a very sudden drop. This means that, with the exception of

5.1. The effect of embedding a line generalization algorithm into the tGAP generation workflow

probably the last around 15%, the data which ends up being transmitted over the internet is very large even when the map should show a very large surface area. There is even a strange rise in the number of cases, which might be explained by a large number of ‘split operations’, which ends up creating even more geometries, and thus nodes.

This can end up making the experience very slow for anyone using the system. Trying to look at what is happening a bit more in-depth, let us take a number of snapshots at some moments of the tGAP generation, and for a particular dataset (Dataset-2, for example). The total number of steps generated by the algorithm in the case of this dataset is always approximately around the 1600-mark. The snapshots will be taken at 1/4 out of the total number of steps, at the middle of the process, at 3/4 out of the total steps, and finally almost at the end (for the purposes of this test, the step 1500 was chosen).

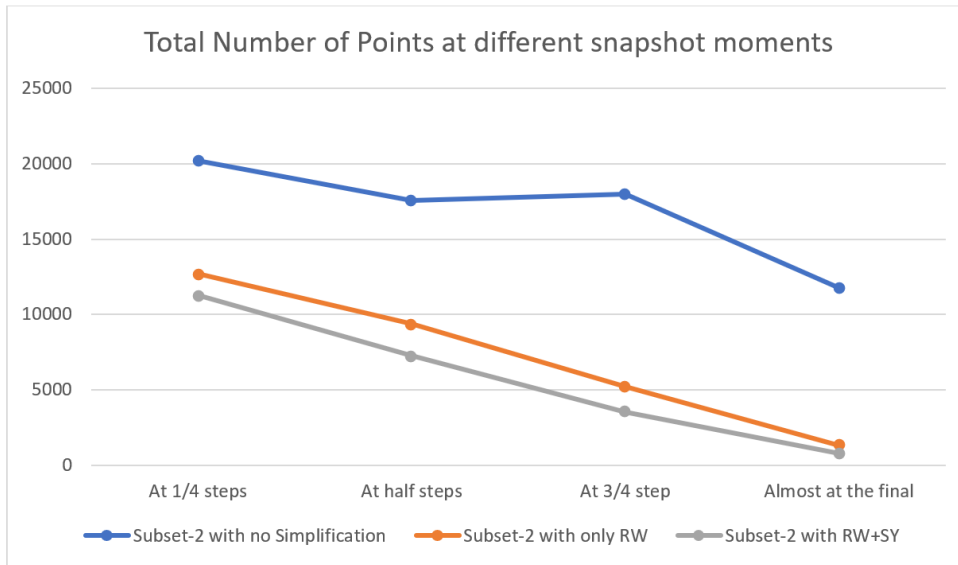


Figure 5.2.: Snapshot of Dataset-2 at four distinct moments in the process

When counting the total number of edges at these steps, we get that, regardless of whether or not we are using or not a line generalization, we have the exact same number of edges across the board (with a variation of maximum ± 10 edges, which I would consider as insignificantly small). However, when summing up the number of individual vertices each of those edges contain, and comparing these values at different snapshots, the contrast is stark (Figure 5.2). Simply introducing a line generalization method into the mix reduces the amount of data, which is a great advantage, when keeping in mind that all of this data should be transmitted over the internet. The contrast is even more visible when looking at the way the map looks, at the last snapshot (because it is less complex than previous ones), in the two different situations 5.3. From this, we can conclude that introducing a line simplification algorithm is indeed a good idea.

Now that it is quite clear that using such a system would be beneficial, when considering the requirements defined at the start of the section, let us now take a look at how this change would influence the look of map objects, from the point of view of the end-user.

5. Results and analysis

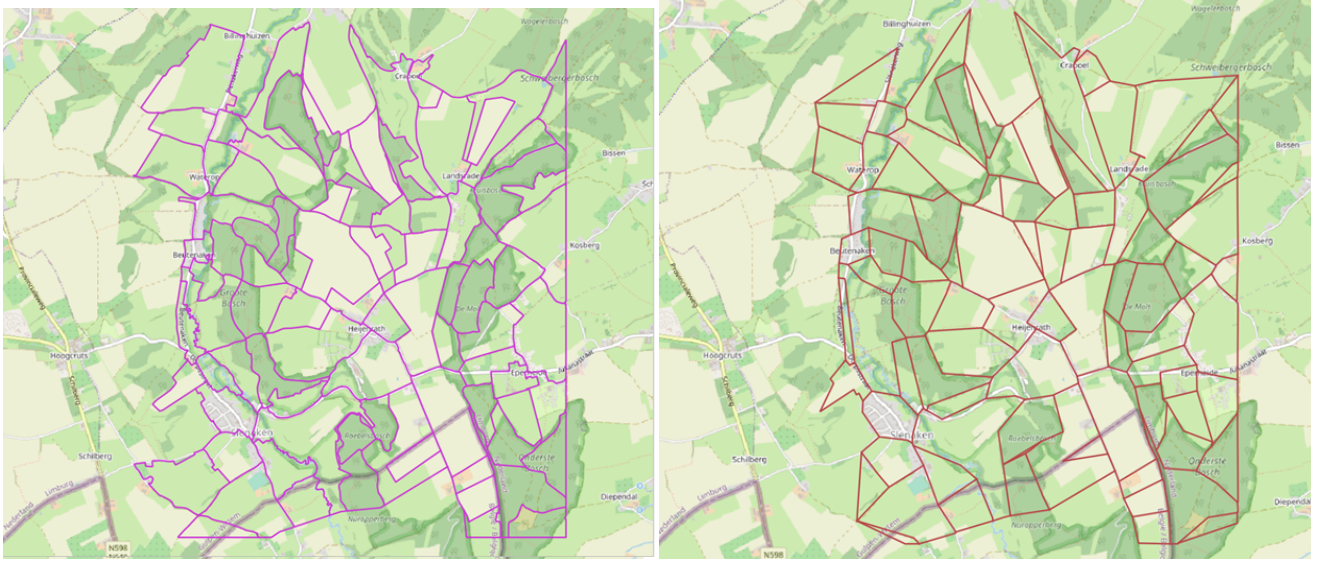


Figure 5.3.: The way the entire Dataset-2 looks at the final snapshot with simplification turned off (left) and on (right)

5.2. Comparison of results from the user perspective

Considering the following two facts: firstly, one of the main focuses of the thesis has been on the implementation of the Samsonov-Yakimova line generalisation algorithm, which was designed with highly angular structured (such as man-made structures or politically-drawn borders); secondly, we may consider that Reumann-Witkam offers good results of simplifications in most other cases. When considering this, let us take a look at how these changes are reflected on the map, from a visual point of view, just as an average user would look at the map, with the spotlight on the geometries which could be handled by SY.

5.2.1. How highly-angular structures evolve in the generalization process

As it was mentioned quite a few times throughout this thesis, the Samsonov-Yakimova algorithm was designed specifically for handling highly-schematic and regular edges. In this category, we can include man-made structures, such as houses, warehouses and so on. Let us take a look at some examples, to analyse the differences between how buildings would evolve (In Figures 5.4 and 5.5). Another example for this situation can be explored in Annex B.1.

As we can see in those two examples, the overall shape is much more improved when using the Samsonov-Yakimova line simplification algorithm. In the case of RW, for Figure 5.4, the simplification results in a triangular shape, while the second example is full of irregularities and strange bends and shapes such as sudden triangular peaks. This is, of course, less than ideal. On the other hand, the SY Simplification keeps the overall orthogonal shape of the building, in both of the cases. An example on how a particular urban area looks like when using both simplifications is presented in Figure ??, at a (relatively-)similar scale (as the rate of change is slightly different, and to better evidenciate the difference, it is not exactly the

5.2. Comparison of results from the user perspective



Figure 5.4.: An example of a situation where Reumann-Witkam results in a ‘pointy’ structure, while the Samsonov-Yakimova algorithm preserves the initial structure

same scale). When comparing the two, it can be observed that SY has the tendency to make everything much more square-ish.

Just as with any other simplification algorithm, there are cases and situations where the result may not be up to expectations, or in accordance to a pre-defined set of rules. In the following section, let us take a look at some examples for these sort of situations.

5.2.2. Situations where SY may be less than ideal

As mentioned when introducing the Samsonov-Yakimova algorithm, it is very useful when it comes to simplifying highly-angular structures. That being said, simply using this simplification module all the time may lead to some less than desired results. Of course, due to the definition of the algorithm, some types of parcel should be excluded from our tests, such as forest area or rivers, as they can end up having a very large degree of highly irregular borders. The only exception will be made for road segments, as it would be interesting to see how such an algorithm reacts to other types of man-made structures from buildings.

For example, while we may consider that most of the buildings have the characteristics which make them suitable for the SY algorithm (such as 90 degrees angles), that may not always be the case. Take, for example, the following structure in Figure 5.7.

In this situation, we have a normal building, which consists of a very circular structure, even though it is categorized as a building. In this situation, when applying the SY algorithm to simplify it, the result is a very strange structure. It loses completely its initial shape, and ends up becoming very unbalanced, with the lower side almost completely disappearing, while the upper getting quite enlarged.

One final thing to look at is how roads would be handled when applying the SY simplification. As these are also structures which were created by man, it would seem like a good

5. Results and analysis

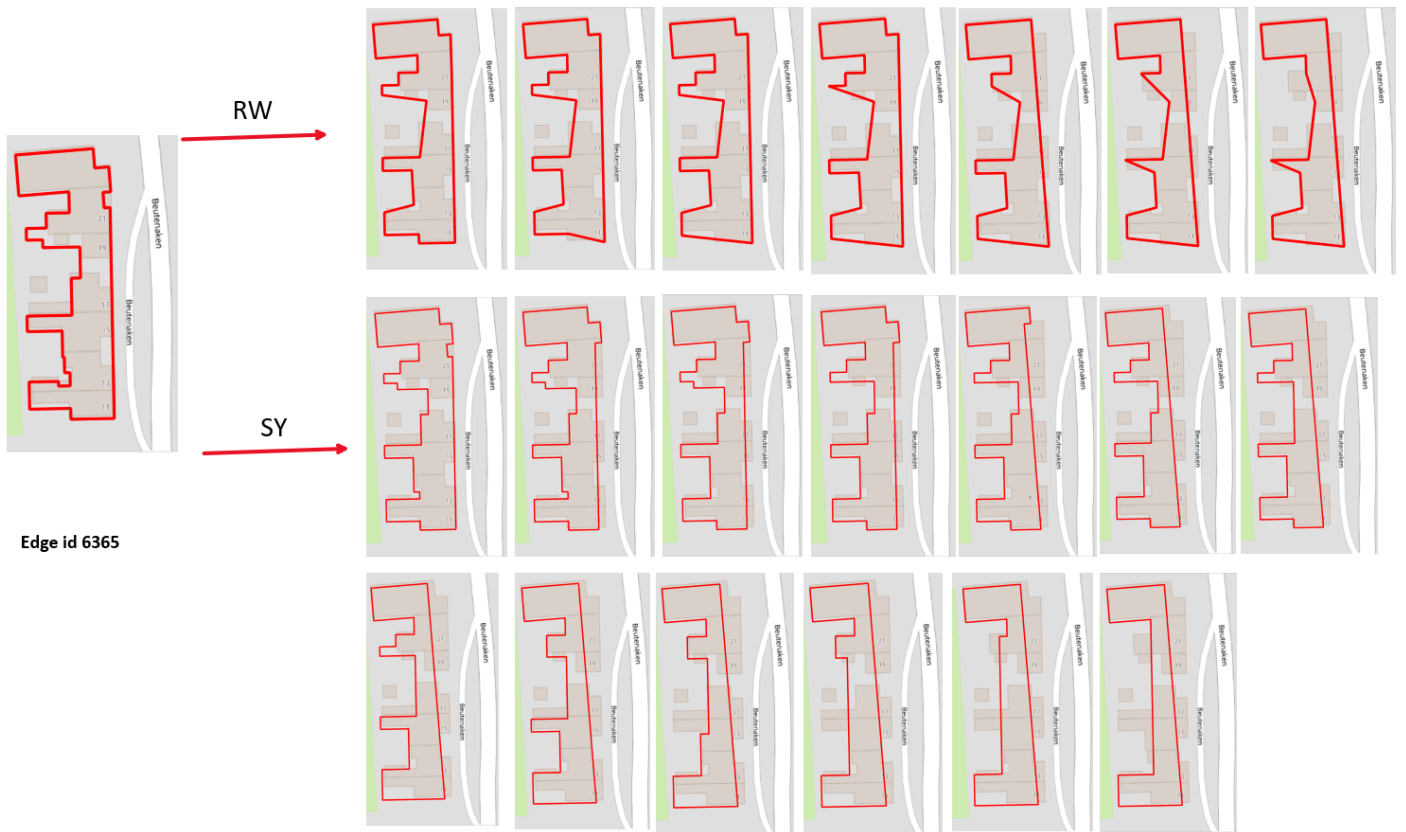


Figure 5.5.: Another example of a built structure

hypothesis to investigate whether or not using this workflow would be suitable for the overall result, from a visual perspective. As we can see in Figure 5.8, the road ends up losing its initial structure. In the latter stages of the simplification, the shape becomes more and more angular, making it look similarly to a structure of buildings. This means that it ends up completely losing its initial features, which makes it almost impossible for a user to distinguish exactly what that shape might be.

For the reasons given above, the following conclusions can be drawn: firstly, it is certainly not a good idea to use the same type of line simplification all the time. The need to categorize what kind of solution is best for which type of geometry seems to be a much better strategy from this point of view, regardless of whether we are discussing about RW, VW or SY. When looking at Samsonov-Yakimova in particular, we can also add to the previous point that, even though a particular structure is man-made, that does not mean that it may be a good idea to use an algorithm focused on angular structures: this is proven both in the case of roadways, railways and such, which almost always end up looking strangely, as well as in the case of round buildings (where they are only a minority amongst all the other objects).

5.3. Consequences of introducing line generalization solutions in the tGAP generation workflow

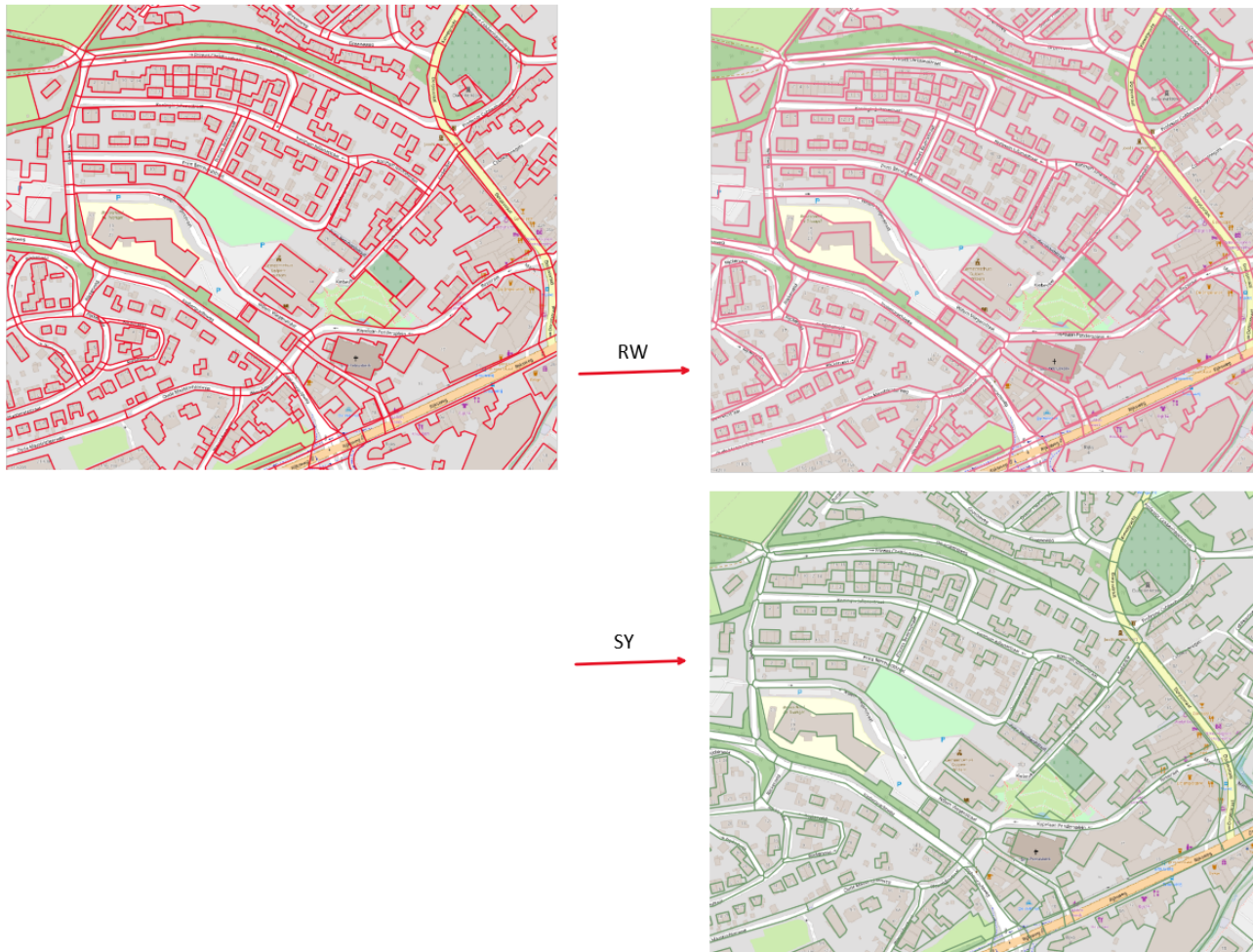


Figure 5.6.: Example of an urban scene

5.3. Consequences of introducing line generalization solutions in the tGAP generation workflow

While the way the map looks is an important aspect for the end-user, at the same time, it is crucial to also analyse how the algorithm itself functions from a technical point of view, and how much the changes which have been implemented affect the functionality and efficiency of the algorithm.

As it was discussed in the [Vario-Scale Maps](#) Section from the Theoretical Background Chapter, the generation of the tGAP structure is only one part in the process of creating a Vario-Scale map. When it comes to the generation of the Space-Scale Cube, we need to keep in mind the fact that its main utilization is done online. For this reason, the process itself needs to be efficient enough to be able to run on an average internet browser, without relying on the processing power of the user's computer. At the same time, the amount of data (which will

5. Results and analysis

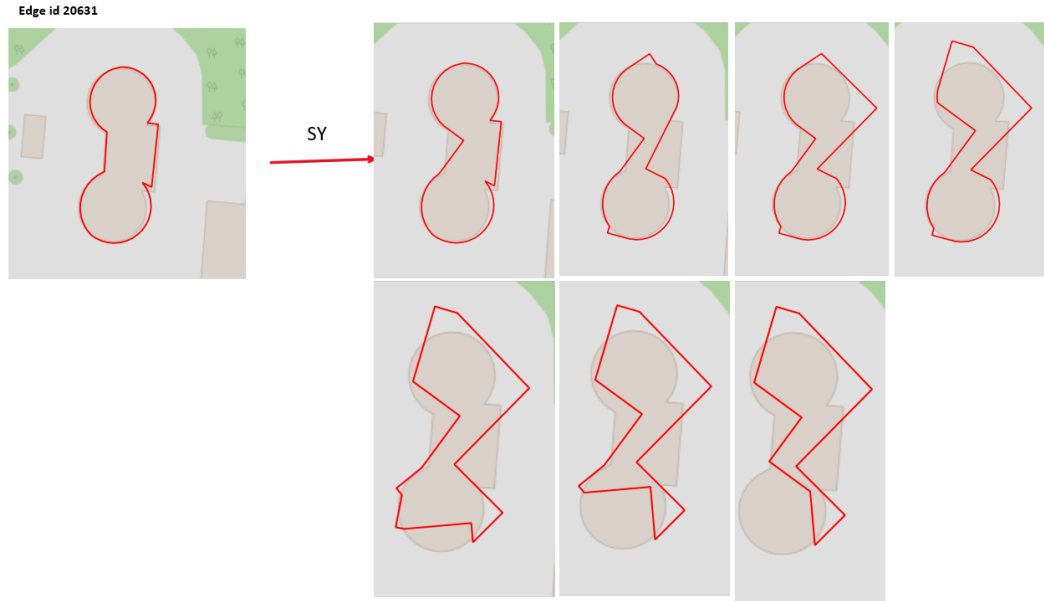


Figure 5.7.: Round Building

be transmitted over the Internet) needs to be taken into account: transmitting a map with too many data points over the web may result in a very bad experience for the end user.

Considering all of these aspects, let us take a look at how the changes implemented in this algorithm affect the overall process of generating the tGAP structure. This section begins with introducing how the different line simplification algorithms affect the functioning of the tGAP generation workflow and is then followed by an insight analysis on the efficiency of the algorithm. Even though the two sections are separate, the analysis itself is performed by taking both into account. For this reason, even though each section explains in-depth their respective topics, the explanation requires mentioning all aspects simultaneously.

5.3.1. Understanding the impact of different algorithms on the run-time

First and foremost, a very good indicator of how well the algorithm works is the time it takes to run: it can give a lot of information, including its efficiency as well as its complexity. After running the code multiple times, with the different simplification methods turned individually on, as well as when using the semantic-based simplification switcher, we get the following results (displayed in Figure 5.9).

When taking a moment to analyse the graph in the aforementioned figure, it is easy to point out how, by simply introducing the Samsonov-Yakimova algorithm into the mix, this has the effect of encumbering the algorithm, and it ends up making it much slower. This is, of course, less than desired.

Trying to analyse the individual graphs from a time-complexity perspective, the two look like polar opposite. In Figure 5.9 (Right), the run-time of the algorithm when using only the

5.3. Consequences of introducing line generalization solutions in the tGAP generation workflow



Figure 5.8.: Roads with SY

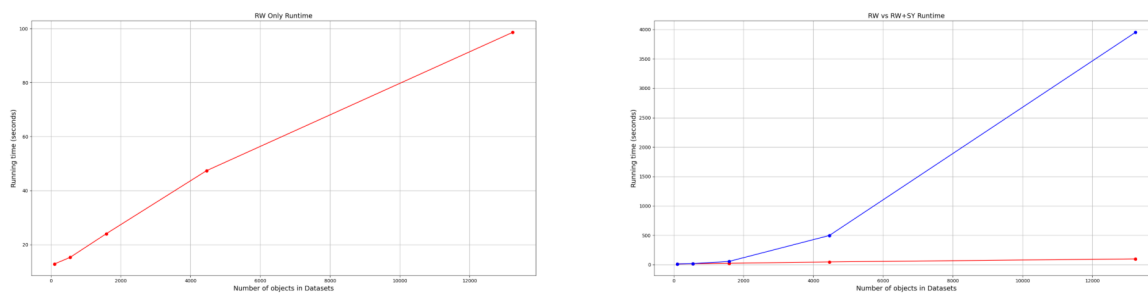


Figure 5.9.: The RW Runtime - standalone (Left) and the comparison between the RW and RW+SY Runtime (Right)

Reumann-Witkam simplification (displayed in red) is barely visible anymore, and looks like as flat line, when comparing it to the algorithm which uses both SY and RW.

When keeping in perspective the fact that the datasets are created with a factor of 3 between them, and when looking at the runtimes of the algorithm when plugging in those datasets, we can draw a couple of observations:

- Firstly, the time complexity of the RW algorithm is clearly Linear (Figure 5.9 Left), as we can see that its run time grows constantly as the amount of data increases ($O(n)$)
- On the other hand, in comparison the the RW evolution, the SY algorithm looks like it ends up working at an almost exponential rate ($O(n^2)$).
- By computing the average time the workflow spends on a face when considering the increasing number of objects per the 3x factor property of the datasets, we can observe a strange trend: while, for RW, it takes on average less time to handle one particular face at a time, for the SY the opposite can be noticed, the amount of time increases.

5. Results and analysis

Based on these observations, it can be pointed out that something is not going right with the running of the algorithm. For this reason, let us look further in-depth at the inner-runings of the Samsonov-Yakimova algorithm, so that we can better understand why all of this is occurring, and perhaps even try to partly mitigate it.

For the purpose of this project, we can define efficiency of particular algorithm as a percentage of how many times it ran successfully (from a total amount of calls done by the workflow of that particular algorithm). Going further in-depth into this definition, we consider that a simplification algorithm has run successfully when it returns a geometry which is simpler (containing fewer components, such as nodes and edges) than the input geometry. Considering these two definitions, the pie-charts in Figures 5.10 show the efficiency of the Samsonov-Yakimova line simplification for highly-orthogonal geometries (The same data, presented in a pie-chart format, can be found in Annex B)

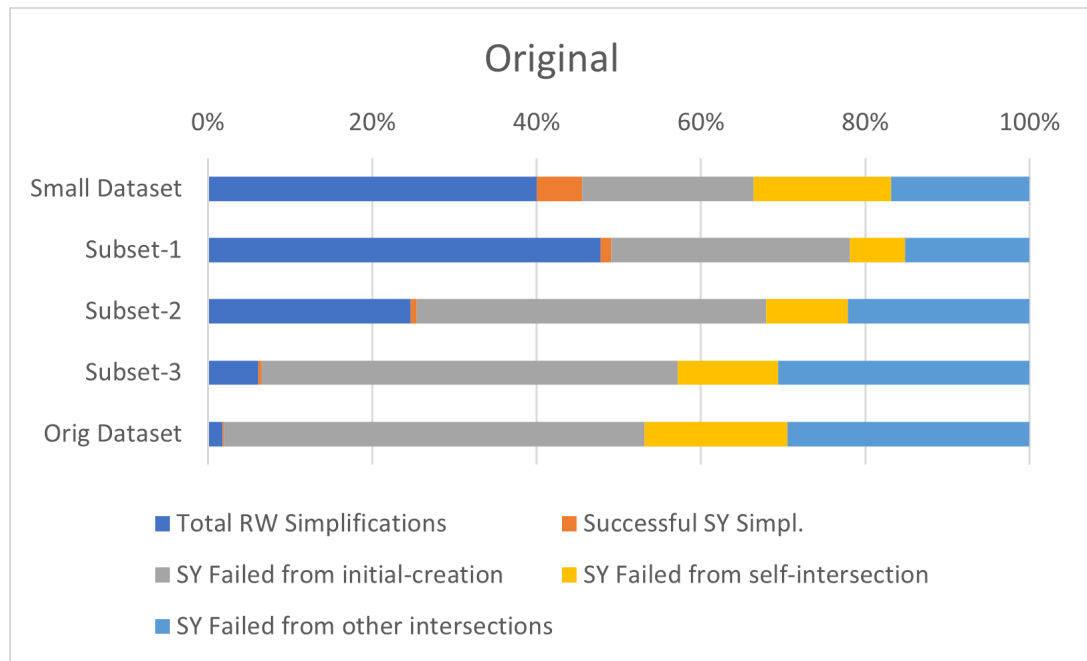


Figure 5.10.: The operations (successfull/unsuccessful) which have been performed - original version

As we can see in the aforementioned [pie-charts], a big chunk of the figure ends up being attributed to SY Simplifications. Looking at the evolution from the smaller dataset to the largest one, it can be noticed that the percentage that the RW Simplification ends up taking gets increasingly smaller. At the same time, another worrying trend that can be noticed is the fact that the SY Simplification succeeds only a very small percentage of the time.

Three main types of failures that start to shape-up as the following, explained alongside the reason why they are occurring and what could be done to fix these issues:

- Failure at initial creation - This occurs when the data structure, which is needed in the simplification, cannot be correctly created. This can be caused by a number of factors, such as the edge containing too few segments (either less than four in the case of a non-circular shapes, or less than six otherwise - so basically simple buildings). In a nutshell,

5.3. Consequences of introducing line generalization solutions in the tGAP generation workflow

this means that the edge does not meet the requirements necessary to for it to undergo the simplification process. A possible solution to this can be to simply keeping track of these problematic edges, and when needing to handle them again at a later point in the workflow, we can decide to either handle them differently (by using another line simplification algorithm, for example), or just skipping them completely;

- Issue in the resulting simplification's structure [SY Failed from self-intersection] - This results when the output geometry of the simplification is complex, and would thus not conform to the structure of the planar partition (an example for this situation may be). This basically means that the simplification process returns problematic structures. This is, of course, less than ideal. One of the causes behind this would be building structures which don't follow the conventional 90 degree structure, which means that the SY algorithm would no longer function correctly, thus resulting in these strange outputs. One solution for this issue would be to try a workaround by temporarily using another simplification, but this has the risk of complicating the shape, and trying to use SY again at a later stage does not guarantee that it will also succeed. A better alternative would be, instead of selecting which simplification to use based on the type of features it neighbours, to do a selection based on the component angles of a particular edge.
- Problem when introducing the object into the planar partition [SY Failed from intersections with other objects] - When working in a tightly-packed planar partition (i.e. where there could be a lot of different objects very close to one-another), there is always a good chance that the simplification process will fail, as the resulting shape may end up intersecting other objects. This is, by far, the most challenging issue that can occur, as the simplification process was successful, yet the change cannot be saved, as it will impact the correctness of the structure. One solution for this would be to keep track of the faces which end up intersecting our simplification. Until those faces change themselves (or disappear completely), there is no point in retrying the simplification, but once that occurs, we can try yet again with our simplification process.

Out of all the causes mentioned in the previous list, the first one can be solved relatively easily, while the latter two would require significantly more complex solutions. Due to the constraints imposed by the nature of such a thesis, namely the fact that the research has to be performed in a limited amount of time, some possible fixes to the last two issues will be presented later in the [Future work](#) Section. The main reasoning behind why it would be easier to solve the first problem rather than the other lies in the particular characteristics to the issue itself: firstly, this is the only situation where the simplification itself is not being called (meaning that the algorithm returns in its inception phase). Secondly, due to the minimum requirements that a particular geometry needs to adhere to such that it can be simplified under SY, there are no changes that we can make, on either that particular edge or its neighbourhood, which would change this outcome (as it is the case with the other two problematic cases, where either slightly changing the geometry (by temporarily using another type of simplification, for example) in the case of self-intersections or by changing or removing neighbouring objects for the intersection with the environment failure).

When applying the changes discussed when introducing the issue, and by replacing the used solution with Reumann-Witkam, we get the following improvements (in [Figure 5.12](#)). With the improved version of the algorithm, the number of failures due to Initial-creation issues end up being only an insignificant part of the entire process. And, while the number of RW Simplifications ends up growing, for each of the cases, with a relatively-constant value, the percentage of SY Simplification Failures caused by intersections seems to take over the

5. Results and analysis

void left behind, with an increasingly larger factor as the size of the dataset increases (where in the case of the Small Dataset, each other remaining running case gets a piece of the puzzle, when it comes to the Original Dataset, the intersection with other objects ends up taking a really large part, thus covering more than 60% of the cases which are being tried in that situation)

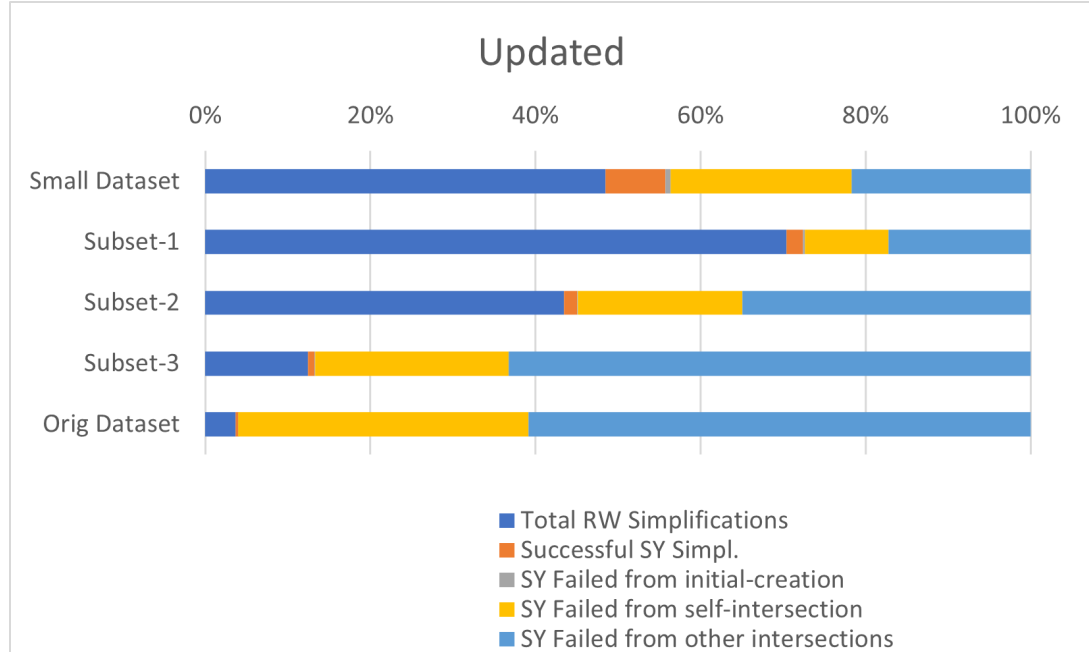


Figure 5.11.: The percentage that each failure/success takes from the total amount of runs - UPDATED

At the same time, when comparing the number of operations (Total operations, with the SY-specific ones as highlighted) that are being performed (in Figure 5.12), it can be noticed that, when introducing this solution, there are always less operations which need to be performed in a runtime. This means that, even though the solution was relatively simple, the impact that it has is quite significant. That being said, while it is an improvement, the overall runtime still remains problematic (Figure B.7 in Annex), which means that further research has to be performed (when introducing ideas presented in the [Future work](#) Section), as to improve this aspect.

5.3. Consequences of introducing line generalization solutions in the tGAP generation workflow

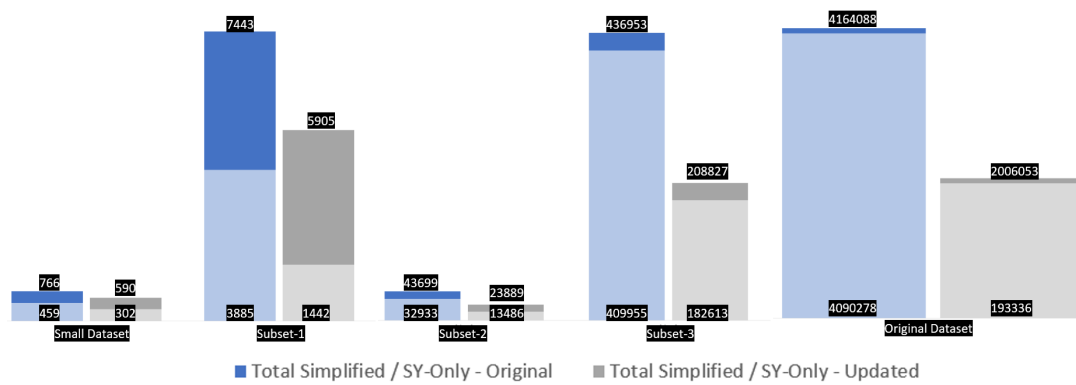


Figure 5.12.: Amount of runs with the original version (blue) and the updated one (grey), for all datasets.

6. Conclusions and Future Work

After analysing the results in the previous chapter, and now that all the pieces of puzzle have been set down, it is time to take a moment and try to briefly go over and summarize all of these concepts by bringing them together. A good way of going about this is by looking back at the [Introduction](#) Chapter, and trying to go over both the main and sub-questions, while at the same time providing an answer for each point. Let us begin with the sub-questions, as to build a solid foundation for then answering the main research question.

6.1. Answers to Sub-Questions

Firstly, let's take a look at the sub-questions referring to theoretical aspects of line generalization

- *Which line generalization algorithms are better suited for which particular situations?*

After looking at both the Reumann-Witkam and the Samsonov-Yakimova Line Simplification algorithms from both the visual as well as the statistical perspective, it is now possible to draw some conclusions about which one is better suited in different particular cases. Starting with the RW algorithm, it is worth mentioning that it is a very useful solution when it comes to most cases, such as all the kinds of natural borders and other relatively-regular features of a map. In contrast, the SY solution when it comes to the highly-regular structures (for example, the footprints of various building quarters, which end up having a lot of almost-90 degrees angles). That being said, it should be noted that there are a number of situations where using RW might be preferred, especially in cases of built structures which do not follow the angularity of the majority of buildings. At the same time, due to the issues which were observed in the SY algorithm design, using RW may also be used as an alternative, unless the algorithm itself ends up being improved in a possible future work, such that (most) of these concerns won't even show up in the workflow. In conclusion, both of these solutions are a good alternative, although SY, in its current state at least, it may end up being less than reliable. Regardless, when putting the environment aside, both have their strengths and weaknesses, but when considering the aforementioned classification, each of them can be used in their best case scenario.

- *What is the most suitable way of combining said algorithms such that it upholds the technical requirements?*

Before answering this question, we need to take a step back and define what exactly "technical requirements" mean. (presented in the [Methodology](#) Chapter), we can then try to find answers to whether or not our solution meets the technical requirements. Firstly, the workflow does run successfully without breaking, which means that we can rule out the first criteria right away. After that, the result of said workflow can be qualified as correct, as the resulting structure does not end up having any sort of topological

6. Conclusions and Future Work

errors. For the final rule, a bit more of a discussion needs to be had. When taking into account the solution for choosing the proper algorithm which has been designed, and looking at the results it spews out from a visual perspective, most of the outputs end up retaining their overall characteristics. That being said, this does not occur all of the time (as discussed in the [Comparison of results from the user perspective](#) Section of the previous chapter), and for this reason we may say that, in regards to this rule, the expectations are not fully met. For this reason, a better fine-tuning of the algorithm would be required, so that the the workflow would end up upholding the technical requirements in their entirety.

Secondly, it is time to answer the sub-questions related to algorithm and data structure design.

- *What are the conditions and the development requirements necessary for maintaining topological correctness at any scale?*

The detailed answer to this question lies in the [Details referring to the tgap integration](#) Section of the [Aspects related to Implementation](#) Chapter. As it was mentioned in the answer to the previous question, the tGAP generation algorithm does uphold all necessary technical requirements, which means that the result of the workflow is indeed topological correct. The way this is achieved is by following the conditions and implementation criteria, which are, in brief: ensuring that all algorithms involved keep track of each-other's individual components, should that be necessary for the good functioning of the workflow (for example, keeping track of the Point geometries in the QuadTree, even though it is used only in RW - situation explained further in depth). At the same time, the various algorithms need to be synchronised and coordinated, both in terms of data structures (where some conversions should be applied) as well as the decisional system which is in place (i.e. the system which decides which algorithm to choose, based on the classification of the neighbouring faces). Lastly, it would be important for all the used algorithms, individually, to keep track of their surroundings so that the structure of the dataset remains conform with our rules. This point, discussed in a lot of detail in the [Integration with the broader tGAP-system and topological aspects](#) Section of the Methodology chapter, refers to the way that individual line simplification solutions ensure the topological correctness of the solution. The factors that maintain the correctness are founded also in the way that newly-created or removed points are handled by the whole system.

- *What is the optimal way of performing operations such that the line/vertices density remains constant, also when taking into account the scale change and its most favorable ratio between the number of objects and the size of the map which is being displayed at that particular scale?*

When reviewing the graphics in Figure 5.1, it would be easy to conclude that simply using our way of handling line generalizations is a very good alternative to keeping the ratio between the density and size displayed of the map. That being said, it might be argued that, in its current form, the SY Line simplification does not do a fully optimal operation, as, per one step, it is able to only do one removal of segments. In this situation, it may be perhaps argued that, in this way, the correct scale may not always be reached. For this reason, it is not possible to guarantee that the shape that a particular object has at a certain scale meets all the requirements, when it comes its epsilon value. In order to make the entire workflow fully robust, it would be necessary to follow either the suggestions presented in the [Consequences of introducing line generalization solutions in the tGAP generation workflow](#) Section from the previous chapter, which refers

to how to improve the shortcomings of the SY algorithm. Alternatively, the suggestions offered in the [Future work](#) Section might also lead up to much optimal results.

- *How can the scale transition be performed in a smooth manner when integrating it into the broader Vario-Scale system? At the same time, what is the best way, from the point of view of time and size complexity (from the perspective of Big O notation concepts [Kuredjian, 2017], when looking conceptually at the efficiency of the various algorithms), to perform line generalization in particular and Vario-Scale operations in general?*

The answer to this question lies again in the suggestions offered in the following manner. Due to the way that the SY runs, it is again quite difficult to uphold the smooth performance requirement. Another way of ensuring that the transition can be performed in a smooth manner would be by simply running the workflow without SY. As it can be seen in Figure 5.9, the linear characteristics of the RW algorithm make it probably the best candidate when it comes to smooth changeovers and mathematically improved time complexity. This being said, the usefulness of the Samsonov-Yakimova algorithm still cannot be negated, as there are some cases where there results it produces are considerably better. At the same time, as we do know the root cause behind the main issues which are occurring when it is used, there is a good chance that, after doing those changes, all of these problems to go away. That being said, for the time being at least, we may not consider it as a suitable answer for this question, and should be considered with mild caution.

Finally, considering the answers to all of these sub-questions, we may move on to answering the main research question:

To what extent can multiple line-generalization algorithms be simultaneously introduced in the Vario-Scale structure such that they preserve the topology and enable an optimal line density (while trying to preserve the characteristics of the initial shape as well).

Before going any further, it could be said that **yes**, it is possible to . That being said, doing such a process may end up being challenging, due to the many bits and pieces that one needs to keep in mind when working out the final quirks when combining those solutions together. All-in-all, in my personal perspective, while there may still be some issues when taking the Samsonov-Yakimova algorithm from its isolation and introducing it into the tGAP generation workflow, the overall integration of multiple generalization solutions still shows quite a few promising signs when it comes to creating a better alternative to the original version.

By using the various coordination processes, which were described both throughout the paper as well as in some of the previously answered sub-questions, the system is able to return a result which does preserve the topology, while at the same time keeping an optimal ratio for the line density. Lastly, besides managing to maintain these terms, the improved version of the tGAP generation algorithm now also manages to better preserve the initial shape of more types of topographic objects, thus giving the end-user an overall better experience. With that in mind, and with most of the research which was done in this paper showing quite auspicious results, it would still be a good idea to also be weary about the performance parameters of the solution, as they are not quite up to the standard.

In the end, no one generalization solution is a silver bullet. They all have some strengths, and some weaknesses. For this reason, the decision of combining them, one way or another, does seem to be the way to go in the future of map generalization automatizing.

6.2. Future work

In the final part of this Graduation Thesis, while keeping in mind some of the shortfalls that we encountered in throughout this research process, it would be nice to present some of the possible things which may be explored in a possible future deepening of this topic.

Sadly, it seems that there may be a number of issues with the methodology itself. The main problem comes from the selection of the edges itself. In the Section 3.2.2, an alternative way of selecting the edges which should be simplified is presented. The solution presented there, while very suitable when used in the context of tGAP generation when using the Samsonov-Yakimova, it can still have the exact same issue when using more than just one simplification algorithms. In a nutshell, different algorithms means different requirements, which in terms means it is more challenging to decide which is the right geometry to be selected for simplification. As a future investigation into this issue, trying to keep track of the multiple ways of selecting the geometries, based on the right algorithm for them, should be researched.

Another problem which can arise from implementing the Samsonov-Yakimova orthogonal line simplification comes from the fact that the algorithm introduces a lot of new geometry: due to this fact, the initially-defined order for the other generalization operations (such as merge and split) may end up changing drastically. While this was the case as well when it came to the Reumann-Witkam algorithm (as a simplification here can also change the size of a particular face in the planar partition), the issue is much more pronounced in SY, due to the drastic changes that can occur. For example, let's consider that we set the SY simplification in such a way that, when it performs a simplification, it ends up enlarging the building objects. In this particular situation, those faces should end up being merged/split at a later point in the process, thus should stay visible for longer on the map. A better bookkeeping of these sort of changes should be introduced in order to solve the issue.

The topic of the Space Scale Cube was not touched at all throughout the duration of this project. For this reason, seeing the ultimate impact that the workflow that was researched in this paper would be another very excellent trajectory for continuing the topic at hand. A good starting point in this direction would be overlaying all of the simplifications which have been performed, then looking, transition to transition, what changes are created, from the most complex version of a particular line geometry to its most simplest. This way, it would be easy to visualise the changes, and use this as a basic for going further into 3D.

Due to the apparently complexity of the implementation of Samsonov-Yakimova (caused by all of the edge-cases which can occur), the development itself can be cumbersome, and difficult to solve issues, should they arise. For this reason, implementing the line-shifting algorithm presented in Buchin et al. [2011] could be an excellent alternative, as it seems easier to adapt for out situations, and has the exact same application as SY. Last thing that would be worth exploring in possible future research on this topic would be a better way of deciding which algorithm to use for a particular situation. For example, instead of looking at the topographic category of the left and right neighbours, we can analyse the geometrical characteristics (such as, for example, by creating a histogram with the values of the component angles). Then, the objects with the same characteristics are grouped together. This idea can go further and further down, with even concepts from Machine Learning that could be introduced (perhaps, in a discussing where we try to teach a system how to detect what the best solution is for individual features).

A. Reproducibility self-assessment

A.1. Marks for each of the criteria

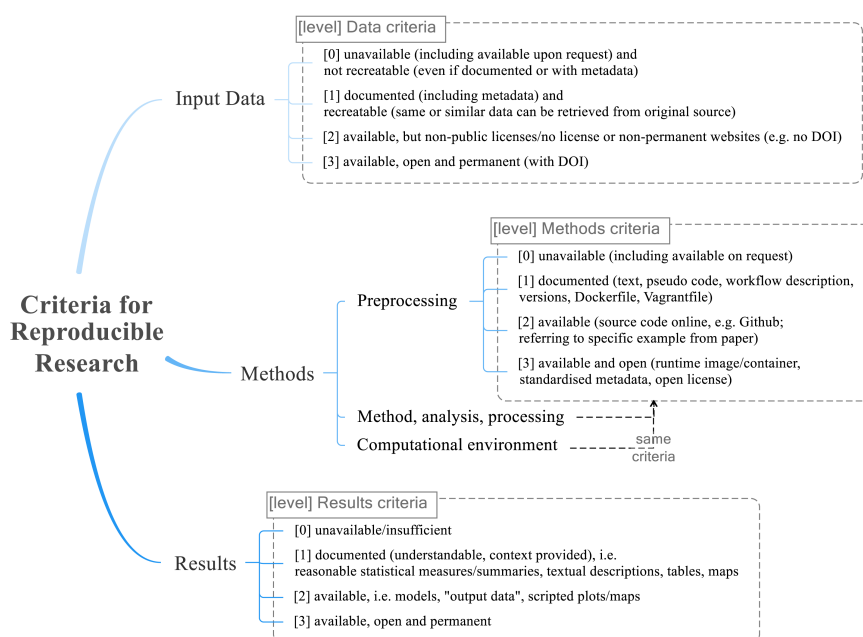


Figure A.1.: Reproducibility criteria to be assessed.

Grade/evaluate yourself for the 5 criteria (giving 0/1/2/3 for each):

1. input data - 3 - All data can be easily accessible
2. preprocessing - 3 - there is no preprocessing needed (if the data is in its correct form)
3. methods - 1 -
4. computational environment - 3 - available on Github
5. results - 2 - needs quite a lot of twaking to reach a final result

B. Various Figures and UML diagrams

Datasets	# Faces	# Buildings	# Edges	# Points in the outline of edges
Small Dataset	98	64	187	1082
Subset-1	546	177	1173	7015
Subset-2	1585	466	3379	21484
Subset-3	4460	1594	9036	54046
Original Dataset	13238	5193	26208	158178

Table B.1.: Compatibility matrix of multiple classes

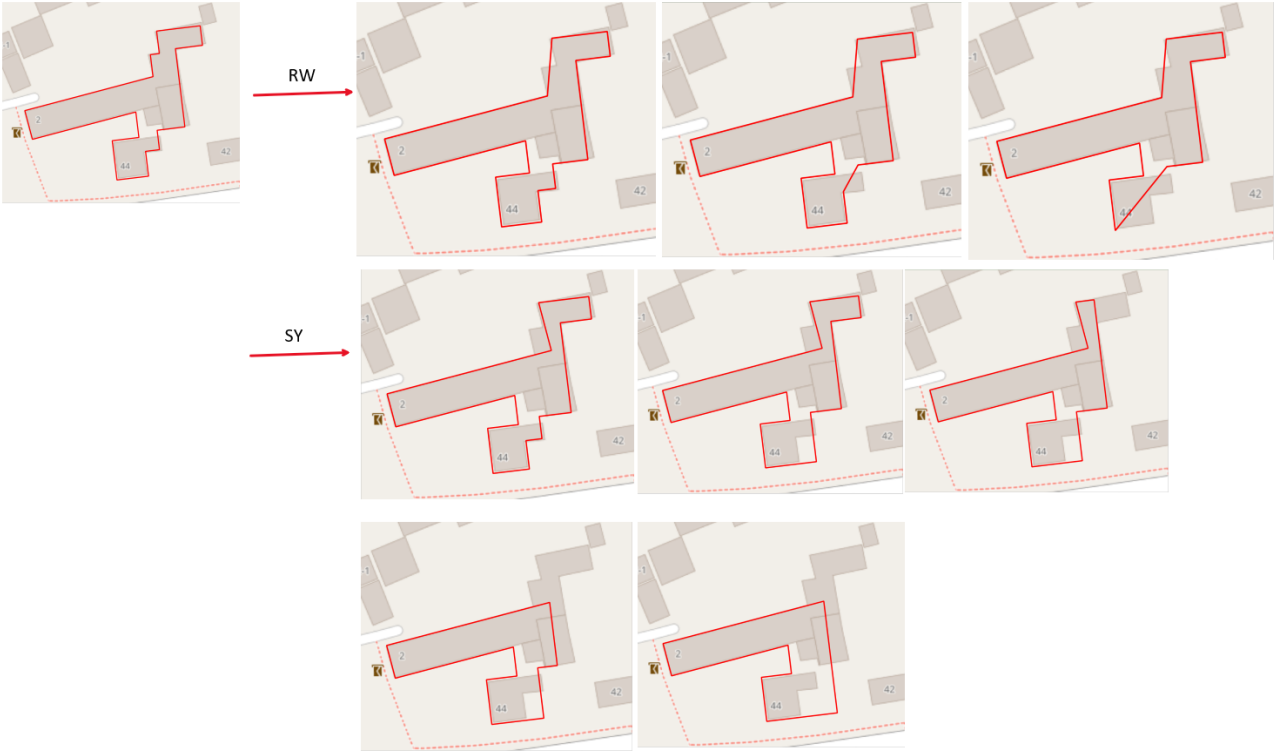


Figure B.1.: Differences between using the RW and SY algorithms on a random building

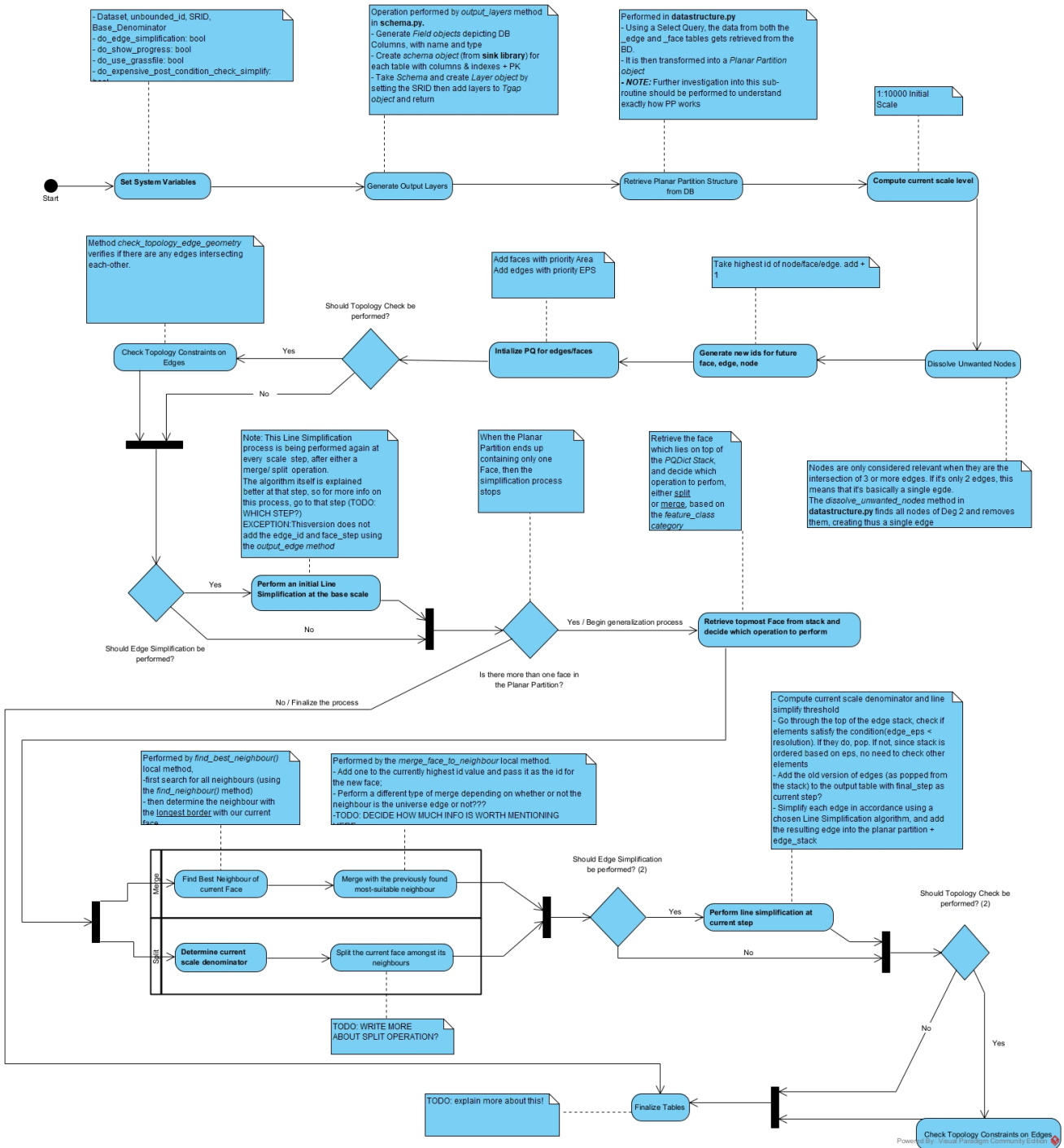


Figure B.2.: Activity Diagram of the tGAP Generation Algorithm

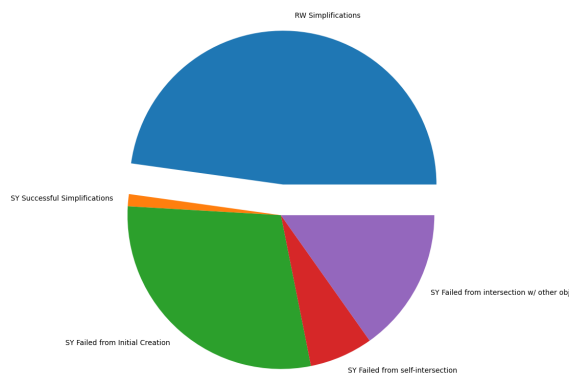


Figure B.3.: Success Status of Simplification calls - Subset-1

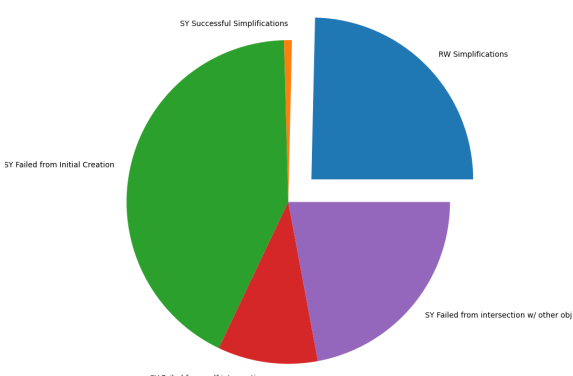


Figure B.4.: Success Status of Simplification calls - Subset-2

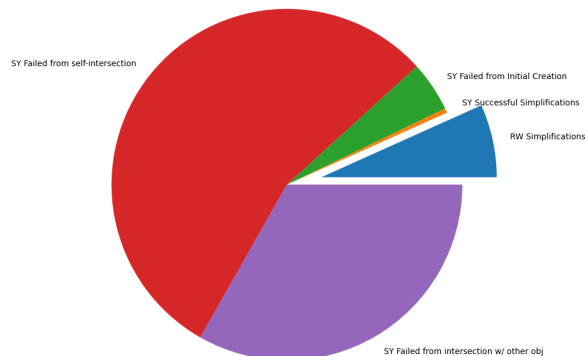


Figure B.5.: Success Status of Simplification calls - Subset-3

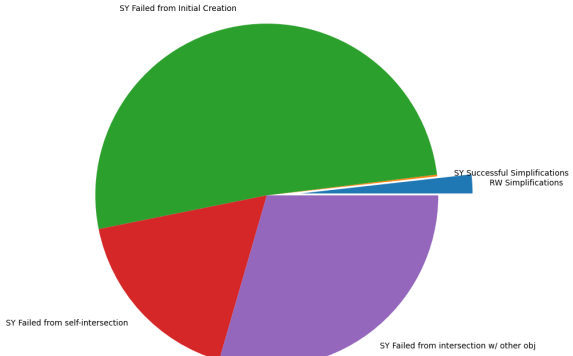


Figure B.6.: Success Status of Simplification calls - Original Dataset

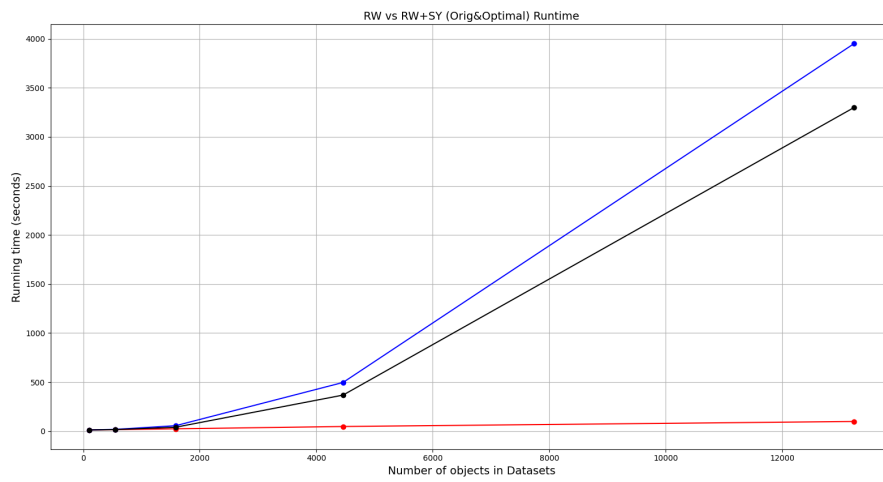


Figure B.7.: Runtime of different variants of the workflow]

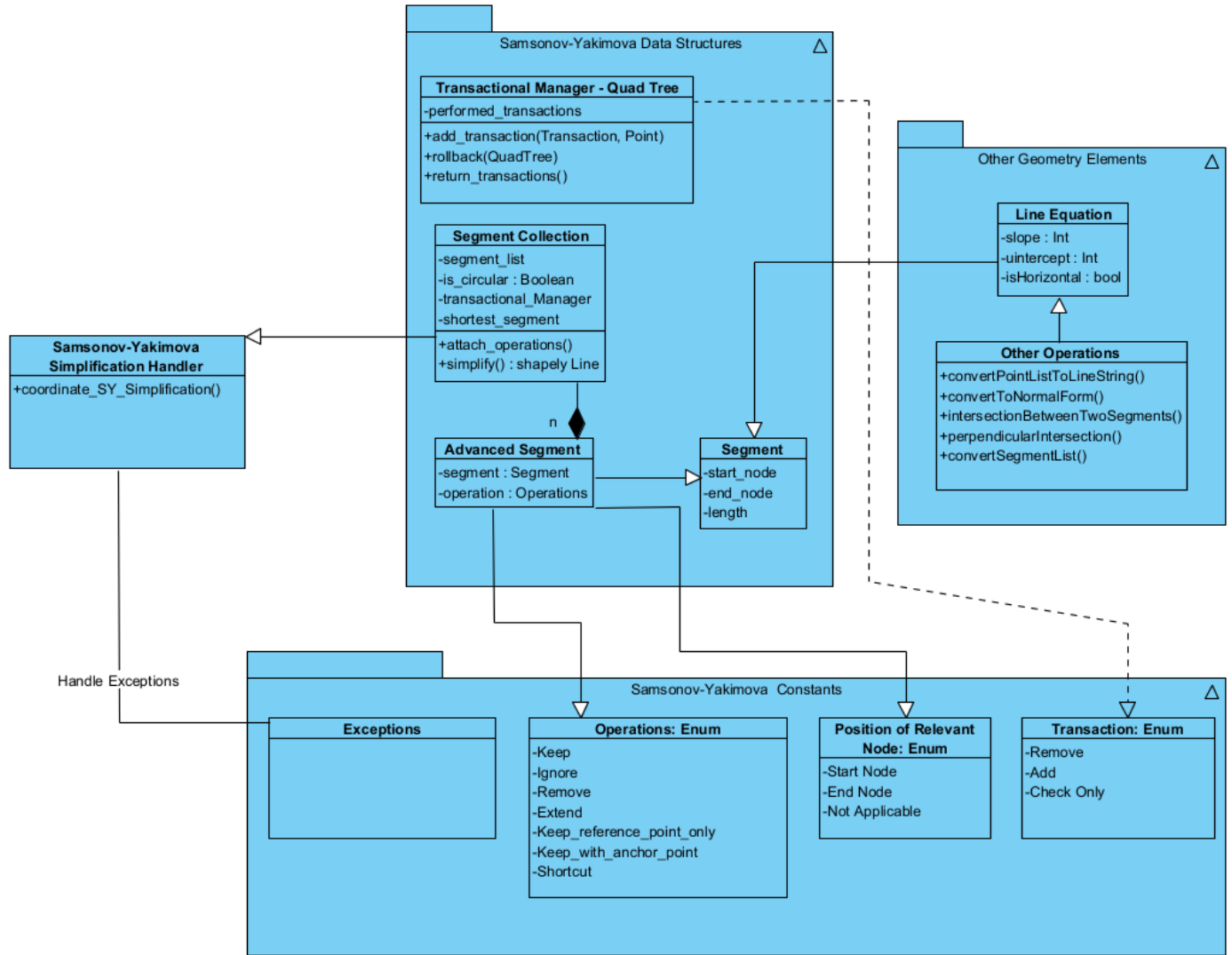


Figure B.8.: The Class diagram for the Samsonov-Yakimova components

C. Various Algorithm - in Pseudocode

Algorithm C.1: Attaching operations to Segment Collection

Data: Segment Collection - *seg_coll*; Index of the shortest segment - *idx*

Result: Segment Collection with Operations attached

```
1 Attach default 'Keep' operation to all segments, and Relevant Node as default 'Not
  Applicable' (NA);
2 for seg in seg_coll do
3   seg.operation  $\leftarrow$  Operation.Keep;
4   seg.operation.relevant_node  $\leftarrow$  RelevantNode.NA;
5 Begin initial length check;
6 if seg_coll.length < 4 OR (seg_coll.is_circular AND seg_coll.length < 6) then
7   return Exception;
8 Shortcut Simplification Case;
9 if seg_coll.length = 4 AND idx - interior then
10  seg_coll[idx].operation  $\leftarrow$  Operation.ShortcutInterior;
11  return seg_coll
12 Begin normal classification process;
13 Go to the right of the idx;
14 try:
15   seg_coll[idx+1].operation  $\leftarrow$  Operation.Ignore;
16   try:
17     seg_coll[idx+2].operation  $\leftarrow$  Operation.Extend;
18     seg_coll[idx+2].operation.relevant_node  $\leftarrow$  RelevantNode.Start;
19   catch IndexOutOfBoundsException:
20     if seg_coll.is_circular then
21       seg_coll[first].operation  $\leftarrow$  Operation.Extend;
22       seg_coll[first].operation.relevant_node  $\leftarrow$  RelevantNode.Start;
23     else
24       seg_coll[idx+1].operation  $\leftarrow$  Operation.KeepAnchorPointOnly;
25       seg_coll[idx+1].operation.relevant_node  $\leftarrow$  RelevantNode.End;
26 catch IndexOutOfBoundsException:
27   if seg_coll.is_circular then
28     seg_coll[first].operation  $\leftarrow$  Operation.Ignore;
29     seg_coll[second].operation  $\leftarrow$  Operation.Extend;
30     seg_coll[second].operation.relevant_node  $\leftarrow$  RelevantNode.Start;
31   else
32     seg_coll[idx+1].operation  $\leftarrow$  Operation.KeepAnchorPointOnly;
33     seg_coll[idx+1].operation.relevant_node  $\leftarrow$  RelevantNode.End;
34 Go to the left of the idx, and perform the same process;
```

Algorithm C.2: Transforming a Median Substitution to a (directionally biased) Shortcut Substitution

Data: Advanced Segment Collection - `adv_seg_coll`

Result: One of the Extend Operations transformed into Keep Only Anchor Point, the point and line elements

```
1 idExtendStart ← get id from adv_seg_coll WHERE adv_seg_coll[id] ==  
  Operation.Extend AND adv_seg_coll[id].operation.relevant node ==  
  RelevantNode.Start;  
2 idExtendEnd ← get id from adv_seg_coll WHERE adv_seg_coll[id] ==  
  Operation.Extend AND adv_seg_coll[id].operation.relevant node ==  
  RelevantNode.End;  
3 if bias is Direction.Left then  
4   | adv_seg_coll[idExtendEnd] = Operation.KeepWithAnchorPoint;  
5   | pointID ← idExtendEnd;  
6   | lineID ← idExtendStart;  
7   | return pointID, lineID;  
8 else  
9   | adv_seg_coll[idExtendStart] = Operation.KeepWithAnchorPoint;  
10  | pointID ← idExtendStart;  
11  | lineID ← idExtendEnd;  
12  | return pointID, lineID;
```

Bibliography

- Hildegard Lewy and Julius Lewy. The origin of the week and the oldest west Asiatic calendar. *Hebrew Union College Annual*, 17:1–152, 1942.
- Georg Gartner. The Relevance of Cartography, 2014. URL <https://www.esri.com/about/newsroom/arcnews/the-relevance-of-cartography/?rmedium=arcnews&rsource=https://www.esri.com/esri-news/arcnews/winter1314articles/the-relevance-of-cartography>.
- Matthew Williams. Imago Mundi, 2019. URL <https://www.jmatthewthomas.com/post/imago-mundi>.
- Daniel Thomas. City of London creates virtual reality map to aid office development, 2020. URL <https://www.ft.com/content/b232d83d-702d-4534-bce6-d10a21d78796>.
- Grand View Research. Digital Map Market Size, Share & Trends Analysis Report By Type (GIS, LiDAR, Aerial Photography, Digital Orthophotography), By Usage, By Services, By End Use, By Region, And Segment Forecasts, 2020 - 2027. Technical report, 2020. URL <https://www.grandviewresearch.com/industry-analysis/digital-map-market>.
- Edie Punt and Jamie Conley. Generalization for Multi-scale Mapping. In *Esri User Conference*, 2014. URL https://proceedings.esri.com/library/userconf/proc14/tech-workshops/tw_585.pdf.
- Anne Ruas. Map Generalization. In *Encyclopedia of GIS*, pages 631–632. Springer US, Boston, MA, 2008. doi: 10.1007/978-0-387-35973-1{_}743.
- Google Developers. Scale, 2021. URL <https://developers.google.com/earth-engine/guides/scale>.
- Marion Dumont, Guillaume Touya, and Cécile Duchêne. Designing multi-scale maps: lessons learned from existing practices. *International Journal of Cartography*, 6(1):121–151, 1 2020. ISSN 2372-9333. doi: 10.1080/23729333.2020.1717832.
- Peter van Oosterom. Variable-scale Topological Data Structures Suitable for Progressive Data Transfer: The GAP-face Tree and GAP-edge Forest. *Cartography and Geographic Information Science*, 32(4):331–346, 2005. doi: <https://doi.org/10.1559/152304005775194782>. URL <https://www.tandfonline.com/doi/abs/10.1559/152304005775194782>.
- Stuart Kuredjian. Algorithm Time Complexity and Big O Notation, 2017. URL <https://medium.com/@StueyGK/algorithm-time-complexity-and-big-o-notation-51502e612b4d>.
- J. C. Müller and Zeshen Wang. Area-patch generalisation: a competitive approach. *The Cartographic Journal*, 29(2):137–144, 12 1992. ISSN 0008-7041. doi: 10.1179/000870492787859763.

Bibliography

- K S Shea and Robert B McMaster. Cartographic Generalization in a Digital Environment: When and How to Generalize. Proceedings Auto-Carto 9, 1989. ISBN 8-89291-209-X.
- Peter van Oosterom and Martijn Meijers. Vario-scale data structures supporting smooth zoom and progressive transfer of 2D and 3D data. *International Journal of Geographical Information Science*, 28(3):455–478, 3 2014. ISSN 1365-8816. doi: 10.1080/13658816.2013.809724.
- Robert McMaster and Stuart Shea. *Generalization in Digital Cartography*. Association of American Geographers, 1992.
- George Merrill Chaikin. An algorithm for high-speed curve generation. *Computer Graphics and Image Processing*, 3(4):346–349, 12 1974. ISSN 0146664X. doi: 10.1016/0146-664X(74)90028-8.
- 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, 12 1973. ISSN 0317-7173. doi: 10.3138/FM57-6770-U75U-7727.
- S Mohneesh. Applications of Ramer-Douglas-Peucker Algorithm in Machine Learning That You Might Not Have Heard of, 2021. URL <https://medium.com/mlearning-ai/applications-of-ramer-douglas-peucker-algorithm-in-machine-learning-that-you-might-not-have-heard-of-1234567890>.
- Ruben Gonzalez Crespo, Lorenzo Romero, Oscar Sajuan Martínez, and Carlos ENRIQUE MONTENEGRO Marín. Design and Modeling to Generalized Linear Elements in a Vector Formatted Cartographic. *International Journal of Advancements in Computing Technology*, 6(3):96–108, 5 2008.
- M Visvalingam and J D Whyatt. Line Generalisation by Repeated Elimination of the Smallest Area. Technical report, 1992.
- Daniel Amigo, David Sánchez Pedroche, Jesús García, and José Manuel Molina. Review and classification of trajectory summarisation algorithms: From compression to segmentation. *International Journal of Distributed Sensor Networks*, 17(10):155014772110507, 10 2021. ISSN 1550-1477. doi: 10.1177/15501477211050729.
- Timofey E. Samsonov and Olga. P. Yakimova. Shape-adaptive geometric simplification of heterogeneous line datasets. *International Journal of Geographical Information Science*, 31(8): 1485–1520, 8 2017. ISSN 1365-8816. doi: 10.1080/13658816.2017.1306864.
- Peter van Oosterom. The GAP-tree, an approach to ‘on-the-fly’ map generalization of an area partitioning. In *Gis and Generalization*, chapter 9, pages 120–132. Taylor & Francis, London, UK, 1995.
- Tinghua Ai and Peter van Oosterom. GAP-Tree Extensions Based on Skeletons. In *Advances in Spatial Data Handling*, pages 501–513. Springer Berlin Heidelberg, Berlin, Heidelberg, 2002. doi: 10.1007/978-3-642-56094-1_{_}37.
- Ken Arroyo Ohori, Hugo Ledoux, and Martijn Meijers. Validation and Automatic Repair of Planar Partitions Using a Constrained Triangulation. *Photogrammetrie - Fernerkundung - Geoinformation*, 2012(5):613–630, 10 2012. ISSN 1432-8364. doi: 10.1127/1432-8364/2012/0143.
- Peter van Oosterom. Spatial Access Methods. In *Encyclopedia of Database Systems*, pages 2681–2681. Springer US, Boston, MA, 2009. doi: 10.1007/978-0-387-39940-9_{_}3639.

- Antonin Guttman. R-trees. *ACM SIGMOD Record*, 14(2):47–57, 6 1984. ISSN 0163-5808. doi: 10.1145/971697.602266.
- Oyewale Oyediran. Spatial Indexing with Quadrees, 2017. URL <https://medium.com/@waleoyediran/spatial-indexing-with-quadrees-b998ae49336>.
- Apostolos N. Papadopoulos, Antonio Corral, Alexandros Nanopoulos, and Yannis Theodoridis. R-Tree (and Family). In *Encyclopedia of Database Systems*, pages 2453–2459. Springer US, Boston, MA, 2009. doi: 10.1007/978-0-387-39940-9{_}300.
- Alexander Kent. Topographic Maps: Methodological Approaches for Analyzing Cartographic Style. *Journal of Map & Geography Libraries*, 5(2):131–156, 7 2009. ISSN 1542-0353. doi: 10.1080/15420350903001187.
- Millie Macdonald. Modular programming: Beyond the spaghetti mess, 7 2020. URL <https://www.tiny.cloud/blog/modular-programming-principle/>.
- Sean Gillies. The Shapely User Manual, 8 2022. URL <https://shapely.readthedocs.io/en/stable/manual.html>.
- Kevin Buchin, Wouter Meulemans, and Bettina Speckmann. A new method for subdivision simplification with applications to urban-area generalization. In *Proceedings of the 19th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems - GIS '11*, page 261, New York, New York, USA, 2011. ACM Press. ISBN 9781450310314. doi: 10.1145/2093973.2094009.

Colophon

This document was typeset using \LaTeX , using the KOMA-Script class `scrbook`. The main font is Palatino.

