Flooding Analysis based on LiDAR Point Cloud Data

Fiona Lippert, Maki Gradecak, Peter Petkanic

Abstract

Flood defence is a topic of major importance in the Netherlands where large parts of the country lie below sea level. To prevent severe catastrophes, a complex systems of dikes has been constructed to keep the water under control. Based on the AHN2 LiDAR point-cloud dataset, we aim at identifying critical regions in the Netherlands that rely heavily on the protection through dikes. This is achieved by generating a digital elevation model, developing a framework for automatic dike detection from raw point cloud data, and combining both into an interactive visualisation that allows to explore the consequences of a dike breach. Facing the massive volume of the given dataset we propose a cloud-computing approach based on Apache Spark and its Python API. Our proposed pipeline is based on distributed point-cloud rasterisation, a variety of image processing techniques and geometric graph analysis.

Keywords — Point Cloud, Digital Elevation Model, Dike Detection, Big Data, Apache Spark

1. Introduction

In The Netherlands, where large parts of the country lie below sea level, it is estimated that approximately 2/3 of the landscape is susceptible to flooding, making the issue of flooding analysis, prevention and control extremely important for its residents. There are a number of methods used for flood control such as dams, floodgates and most predominantly utilised in The Netherlands, dikes.

A dike is a raised ridge that is constructed parallel to the river or shore. In situations where the water level adjacent to the dike raises unexpectedly, the dikes prevent the water level from spreading to the area on the other side, thereby preventing destruction of land and property.

Located across The Netherlands are large systems of connected dikes, some natural, some man-made, providing effective flooding control. However, what would happen if one of those Dike systems were to be compromised, be it through natural erosion, natural disasters or human intervention?

When considering disaster situations such as flooding, it is important to not only evaluate the effectiveness of preventative approaches but also to consider the impact of these approaches failing.

In that regard, this project targets 3 main goals. To create a *coarse grained elevation map* of The Netherlands, to develop a *framework for automatic detection of dike systems*, and using these two data products, to create an *interactive flooding simulation* predicting the flooding zone if these dike systems were to fail.

Developing such complex products on the scale of an entire country is generally an unattainable goal without various rich data sources to drive the analysis and simulations. However, with the increasing amount of geospatial data being collected and made openly available, projects such as this become increasingly possible. The raw geospatial data sets come in a variety of formats, depending on the purpose and acquisition technique. Among them, point-cloud data in LAS format is extremely rich in information and can be used to model accurate 3D surfaces and to extract complex spatial features.

The disadvantage of this data boom however, is that it is no longer feasible to operate on this data using single commodity or even high performance computers, as the data volume often spans into the order of terabytes. Thus, there is an emerging requirement for efficient big data processing techniques for working with geospatial data in order to truly benefit from it. With this in mind the aim of our project is not only to investigate potential flooding hazards, but also to investigate various techniques for efficient processing of large volumes of geospatial data.

To accomplish our task we will utilise the Apache Spark framework[1] for large scale data processing.

The dataset we are working with is the AHN2 dataset which consists of approximately 10 Terabytes of point-cloud data, effectively mapping the entirety of The Netherlands to a 20cm resolution. In order to be able to store and even process such a large amount of data we will deploy our processing pipeline on the Surfsara Hadoop cluster, consisting of 170 compute nodes totaling to a 1370 CPU cores, backed by 2.3 Petabytes of storage capacity.

The remainder of the paper is laid out as follows. In Section 2 we will discuss work related to our own. Our research questions will be presented in detail in Section 3. Section 4 provides further insight into the data that will be used as a source for this project. We will present the setup of our Project in Section 5 where we will discuss how we accomplished our goals. Section 6 evaluates and discusses the results of our experiments. Finally in Section 7 we present our conclusions.

2. Related Work

Our work falls into three categories, flooding assessment in The Netherlands, geographic information science (GIS) and massive point cloud processing using cloud computing. In the following we present relevant related work from all of these areas of research.

2.1. Flooding Assessment in The Netherlands

With flooding being an omnipresent danger, flooding analysis in The Netherlands is a well studied area, with many different initiatives such as the national *Delta Programme* [2] aiming at optimized flood defences, or the *Floris* project [3] exploring topics like risk analysis, loss of life, cost benefit analysis, uncertainty modelling etc. The main starting point for such studies is the identification of potential weak links in dike rings, which may be caused by overtopping or loss of stability due to erosion, sliding or piping [4], followed by a complex analysis of consequences.

For an accurate identification of regions being affected by a potential dike breach simulations are used to predict flooded areas. Such simulations range from simple intersection of the water level plane with a surface model of the landscape [5] to sophisticated 3D solutions based on the Navier-Stokes equations and turbulence modelling [6].

Our work differs from the described ones in that our analysis is not based on existing terrain models and mapped dike systems, but rather aims to automatically generate these from raw data and perform basic flood plain predictions. In that sense, our project can be seen as ground work that transforms available data sources into products to be used in higher level flooding analysis.

2.2. GIS Methodology and Tools

This leads us to the topic of GIS technologies that facilitate raw geospatial data processing and generation of geographical models. Methods used for transforming 3D points into a digital elevation model (DEM) can be grouped into two stages. First, points need to be filtered to separate ground points from non-ground points. Since ground point features depend strongly on the terrain type, a variety of approaches has been studied and evaluated on diverse terrains. Lowest elevation sampling [7] is a simple initialization procedure that assumes that bare earth points are usually the lowest features in a local neighbourhood. More sophisticated ground filtering approaches build on weighted linear least squares interpolation developed by Pfeifer [8], slope-based filter developed by Vosselman [9], or filtering based on mathematical morphology [10]. Second, the resulting points need to be combined into an elevation model. Two model types dominate the research in this area: raster DEMs which can be searched and analysed by simple kernels, and so called Triangulated Irregular Networks (TIN) which allow for accelerated search for neighbouring points [7]. These models are created by interpolating between the irregularly spaced ground points. Here, one of the most popular interpolation methods is Inverse Distance Weighting which uses the weighted average of points in a neighbourhood, where points closer to the location of interest dominate [11].

Identification of ground-points as well as automatic dike detection falls into the category of geospatial classification. Here, two main approaches can be found in literature: classification based on raw point features, and image classification based on rasterised data. Bao et al [12] present an interesting approach for separating ground and vegetation in point-clouds by analysing skewness of the intensity distribution.

In general, a variety of powerful GIS software libraries is available, such as ArcGis, QGIS and Point Cloud Library (PCL), that provide tools for conversion, rasterisation, segmentation, object identification, spatial indexing and more. However, they are designed for standalone environments only and are hence not feasible for large-scale analysis.

2.3. Large-Scale Point Cloud Processing

In contrast to the well-established point-cloud tools for single machines, only little research has been conducted on massive point-cloud processing using distributed computing technologies [13].

Apache Spark frameworks for geospatial data such as *GeoSpark* and *Geotrellis* provide functionalities for spatial RDDs of raster and vector data providing special SQL queries such as distance based joins as well as efficient spatial indexing. However, they do not come with any LAS point-cloud support. To our knowledge, the only Spark library designed for LAS file import is *IQmulus* which relies on an outdated version of Spark. Facing these challenges, several designs of efficient frameworks for distributed point-cloud processing has been proposed in the past years. Boehm et al [14] utilise an Apache Spark based approach similar to our own. By encoding a list of point cloud files as an RDD and distributing it amongst workers, point cloud segments can be individually read and processed concurrently. Wang et al [15] take a different approach to point cloud processing, by using a MapReduce based approach. While this approach proves to be efficient for single transformations, repeated operations on the point cloud can prove challenging due to the limitations of the MapReduce framework. Liu et al [13] provide the specification and implementation of a Spark library for ingestion of large point clouds. While our initial data exploration was based on the library produced by the authors, our final solution moves away from this approach due to technical incompatibilities, but utilises the core concept of distributing the loading of data to many workers.

3. Research Questions

The main goal of this research is to summarize the Dutch terrain in form of a DEM and to detect a topology of dike systems that protect critical areas below sea level. These goals will be achieved based on the AHN2 point-cloud dataset, which imposes major challenges due to its sheer volume. With this in mind, we formulate the following questions to be answered:

- How can we efficiently load and process the point-cloud data on the Surfsara Hadoop Cluster, transforming the data into a workable format and reducing it to a feasible volume?
- How can we transform the unstructured point records into a DEM that appropriately represents the regional ground levels?
- Is it possible to develop a framework for automatic detection of dike systems that is solely based on point-cloud elevation data?
- Can we, based on the DEM and the topology of dike systems, make predictions about scenarios in which one or multiple dikes breach? Which areas would be flooded and how would these flood plains change with rising sea level?

The results are to be summarised into an interactive visualisation which facilitates exploration of the Dutch terrain, the detected dikes, as well as predicted flood plains.

4. Data

The AHN2 dataset is a large 3D point cloud scan of the entirety of The Netherlands, collected over a number of years using airborne LiDAR technology. By utilising lasers, and calculating the time taken for a projected laser beam to return, in combination with GPS positioning, distance data can be inferred and highly accurate X, Y, Z points of the ground as well as features such as vegetation and buildings are extracted. This highly detailed elevation data has many uses, from generating accurate 3D models to detecting abnormalities on the ground, fig 1 shows an example visualisation of such point cloud data.

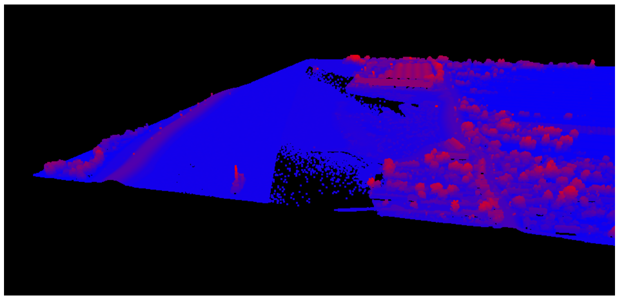
Figure 1 A point cloud visualisation of a sample file from the AHN2 dataset

The entire dataset consists of billions of 3D points, resulting in a total of over 10 Terrabytes on disk space. The data is organised into 225 tiles, each approximately spanning a 6.5 x 5 km area, with each tile further divided into a variable number of data files. The x and y co-ordinates translate to the Amersfoort co-ordinate system, which is one of the reference co-ordinate systems used in The Netherlands [16].

The average density of measurements is about 10 points per $1m^2$, ranging from detailed descriptions of buildings with hundreds of points per m^2 to very sparse representations of water bodies.

A major challenge of this data set is the fact that the points are completely unclassified. In contrast to the recent generation AHN3, there is no distinction between ground, water, trees and buildings and therefore points must be classified or filtered before they can be used in order to avoid distorted results.

For the purposes of this project we are only concerned with the ground level. Performing a quick analysis of some data samples, fig 2 shows that the distribution of height values is highly skewed, indicating a clear distinction between ubiquitous ground points and small-scale landscape features with varying heights. However, the points relevant for dike detection are expected to lie in the same histogram range as irrelevant objects like medium height vegetation. We will come back to this issue in Section 5.

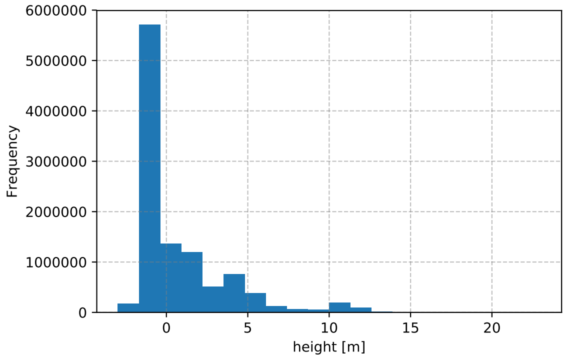

Figure 2 Distribution of ground point heights in a sample of the AHN2 dataset

The second big challenge in working with this particular dataset is the format in which the data is provided. The LAS format is an open format for LIDAR data containing binary data split into a header block, a meta-information block, and the actual point data record [17]. However, storing 10TB of raw data in practise is not feasible and therefore the data is compressed using a special compression algorithm optimised for point clouds, leaving us with 1.6TB of raw data in compressed LAZ format. While this might sound like a big advantage, as we will see, the compressed format is more difficult to work with and furthermore the extra decompression step using the LasTools suite adds a non negligible delay in processing when the data is on this scale.

5. Project Setup

The envisaged data product, as well as the data used to drive the project are quite large in scale. With this in mind, we split the process into 5 stages. In stage 1 - *Preprocessing* we will tackle the challenge posed by the massive size of the dataset, and rework it into a more usable format. Building on this, in stage 2 we create a *Digital Elevation Model (DEM)* of the terrain using the processed data from stage 1. Following on from this in stage 3 we aim to extract dike segments as polygon information by detecting areas of increased height surrounding bodies of water. From there, in stage 4 we run a connected component algorithms in order to detect the dike segments that belong to a single dike system. And finally in stage 5 we aim to bring the data generated from stage 2, 3 and 4 together in a Browser based interactive simulation.

5.1. Preprocessing

The resolution of the points in the AHN2 dataset is far higher then what is realistically required in order to reliably achieve our goals. While larger resolution may at first sights appear beneficial, in reality the increased data size and its associated computational costs outweighs the benefits of the increased richness of detail. Moreover, the given point cloud data is completely unstructured and needs to be transformed into a cleaner data structure.

With the above in mind, we chose to reduce the data to a 2D raster with a $1m \times 1m$ resolution. This means that all measurements falling into a particular $1m^2$ sampling window are represented by one single point on the raster, where the window size is chosen such that it facilitates efficient processing while still retaining sufficient information for dike detection. With the aim of identifying the regional ground level, we implemented minimum elevation sampling by aggregating points within the window and assigning the minimum of all height values in the point group to the respective raster point, discarding undesirable small-scale structures with higher elevation. Knowing that the resolution of the original unstructured data is approximately 10 points per $1m^2$, this rasterisation process allows us not only to project the measurements onto a regular 2D grid space but also to reduce the amount of data by almost factor 10.

The sampling of the tiles is an inherently parallel problem, as each tile is composed of on average 300 files which can be sampled individually without the need for explicit co-ordination, thereby reducing the network overhead. With this in mind Spark was the preferred framework of choice. A list of files inside of a tile is first converted to Spark's Resilient Dataset Distribution(RDD) format. The RDD is then partitioned and distributed across the workers in such a way that each worker task involves reading on average 2 files ($partitions = num\_files/2$). Subsequently a *flatMap* operation is invoked on the RDD such that each file name is converted into a list of sampled points. The resulting list of lists is then flattened.

The workers *map* task has a degree of complexity to it for a number of reasons. Primary complication being the compressed point cloud files that it must be able to read. While Laspy, the tool we are using for reading our point cloud data has compressed file support, its compression support is implemented in a rather unusual way. In order for compressed reading to be successful the worker must have a binary called LasZip (part of the LasTools suite [18]) placed somewhere on its PATH. This external dependency makes the worker task more complicated as the LasZip binary must be recompiled to match the linux architecture that the workers run on and be shipped to the worker nodes when the spark job is submitted. With this out of the way, upon receiving the task, the worker looks up the file in HDFS and downloads it to his local temporary directory. Here the file is decompressed and loaded into memory where it is sampled using the previously outlined procedure. Note that the sampling process is solely implemented through Spark SQL queries working on the point RDD. The overall structure of the preprocessing stage is illustrated in fig 3.

On completion of stage 1 we effectively reduce the data volume from 10TB to 150GB. Bringing the size of individual tiles down to <1GB per tile, on average.

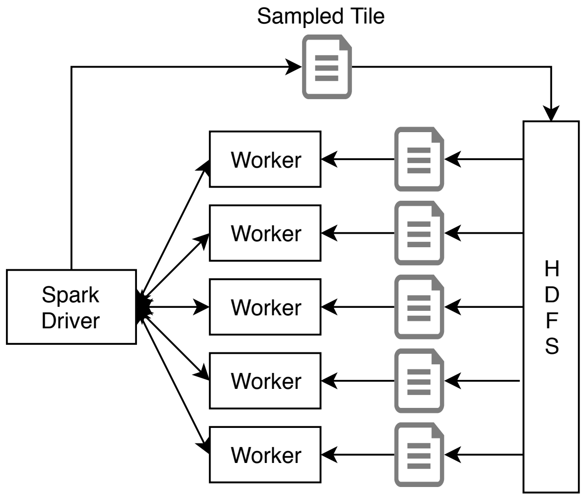

Figure 3 Overview of the sampling architecture, Parallel task execution is driven by Spark framework, with HDFS being utilised as a datastore. Each worker is issued a task of ingesting point cloud files. While ingesting data is sub sampled to reduce resolution and the resulting RDD is written back to HDFS

5.2. DEM Generation

Generating the DEM follows from the points sampled in the preprocessing stage. The minimum elevation sampling constitutes already the first step of DEM construction as it filters out non-ground measurements and it yields evenly spaced grid points. Facing the lack of Spark point-cloud frameworks, we refrained from implementing complex interpolation-based methods from literature which would require computationally expensive repetitive searches for neighbouring points in the unstructured dataframe. Instead, we opted for an image processing approach, where raster points are inserted into a Python *numpy* matrix on which kernel convolution can efficiently be applied in order to obtain raster elevation data representing a smooth surface. We claim that, for our purpose of generating a coarse grain elevation map of the Netherlands which provides information about areas being susceptible to flooding, it is not essential to create an highly accurate surface model including information about local terrain slopes; but a mapping of medium scale regions to discrete height levels is sufficient. Inspired by the Inverse Distance Weighting interpolation approach, we decided to base the definition of these regions of equal height on a the application of a Gaussian filter kernel which essentially sets the value of each raster point to the weighted average of its neighbours, favouring the nearest ones [19]. This reduces noise due to small-scale features that deviate from the general ground level of a region and that had not been discarded through minimum elevation sampling.

To obtain a model that represents areas of equal height, we define a number of height bins spanning the entire elevation range of the Dutch terrain ($-50m$ up to $300m$). These bins are unevenly distributed, such that heights close to sea level, which are more important for flooding analysis, are represented in more fine grained steps than larger elevations.

Due to the fact that the tiles obtained by stage 1 are still quite large in size, generating a DEM outright from an entire tile is not only unfeasible, but also means losing out on the high level of parallelism facilitated by using Spark. Therefore, each tile is first subdivided into spatially contiguous blocks of 500x500 points using Sparks Dataframe *groupby* functionality before the actual processing is done, allowing us to utilise the cluster once again for processing.

Following this, a user defined function (UDF) is applied to each of the blocks. Each worker can apply the UDF on a block independently of the other workers, making this problem embarrassingly parallel similar to the sampling problem in the preprocessing stage.

The process for transforming a chunk of X,Y,Z points into a partial DEM is implemented as follows. Initially, each worker transforms the chunk into a *numpy* matrix in order to benefit from a number of efficient matrix operations and image transformations that Python's *numpy, scipy* and *scikit-image* libraries offer. Next a downsampling function is applied to reduce both the amount of computation per block and the size of the final overall DEM. During the downsampling first a Gaussian filter is applied to avoid aliasing effects, and afterwards a block of 10x10 points is merged into one by extracting the median height value. Following this, another Gaussian filter is used to obtain a smooth image representation of the ground surface. Finally, the heights are binned into discrete values, and the resulting DEM is returned to the Spark Driver. The driver, inserts the returned blocks into an image representing the entire tile and the discrete height levels are mapped to RGB colours, resulting in an elevation map in PNG format.

Note that due to the distributed processing of individual blocks artifacts at the block edges may occur. A more accurate approach avoiding such artifacts would involve halo exchange from neighbouring blocks such that smoothing of edge points would consider their actual neighbours in the overall picture instead of approximating their values using the edge values them selves. However, this would imply significant communication overhead and thus counteract Spark's notion of data partitioning to facilitate fast querying.

Finally, we apply the same approach used to generate the raster DEM to extract a vector representation of areas of equal height, which will be easier to incorporate into flood prediction. The only difference is that for this purpose we operate on the scale of the entire dataset, which means that all sampled tiles are loaded at once and more radical downsampling is applied to allow for concurrent processing of the entire dataset. The resulting raster DEM is separated into collections of areas at 8 distinct height levels close to sea level, represented by 8 binary images. Then for each level ordered lists of contour points are extracted which are transformed back to the original Amersfoort co-ordinates, and converted to a string representation of polygon objects in Well-Known Text (WKT) format, a markup language widely used for representing vector geometry objects through their coordinates. Being of regular string type, these objects can then be collected into a Spark DataFrame at the Spark driver, which can easily be converted into a GeoPandas dataframe containing all identified polygons. The vector DEM in form of a set of polygons is then written to GeoJSON, a format commonly used to encode geographic data structures in form of geometric objects, and thereby presents a concise and workable product which can easily be used for further analysis and visualization.

5.3. Dike Detection

The dike detection process begins similarly to DEM generation, the same general approach is adopted where sampled tiles are divided into blocks, distributed among workers which then apply image processing techniques to their chunk of data. The main difference is that for reliable detection of dike segments a larger area of operation is needed in order to distinguish between long but thin dikes and other small features of similar height. Therefore, the tile is first further downsampled in its entirety by grouping 10x10 areas of points and retaining only the mean height value. The newly sampled tile is then divided into blocks of 500x500 points, representing physical regions of 5km by 5km.

To identify dike points, we decided to follow an image classification approach based on edge detection and segmentation. A dike is a raised ridge with a crest between 2 and 5 meters width and with slopes to both water side and protected land side, resulting in a base width of up to 50 meters. Depending on the dike type, sea dike, river dike or lake dike, the crest is usually between 5 and 10 meters above the surrounding ground level [20]. With this knowledge, we expect dikes in the raster image to appear as bands of points with increased height value exhibiting a strong gradient alongside both edges. The discrete Laplace filter [19], approximating the second spatial derivative, is able to detect exactly these areas of strong intensity change together with the orientation of gradients. Hence, after Laplace filtering the dike edges manifest as pixel values around zero, whereas pixels close to the edges take negative values if they are on the higher side of the edge, and positive values if they are on the lower side. With the dikes being long but narrow structures, it is reasonable to assume that the entirety of the dike appears as negative pixel values, forming segments that can be extracted by simple thresholding.

From the technical perspective, a new UDF for dike detection is used to perform a transformation operation on the grouped data chunks. As before, a worker starts by transforming the assigned chunk into a matrix. Listing 1 describes the process of identifying potential dike segments in such a chunk. The first step is to identify the ground level Z_G. By generating a histogram of heights for the given matrix the most frequent height bin is selected as the ground level. Following this, all of the heights in the grid are clipped to the range $[Z_G, Z_G + 5]$, mapping all heights below Z_G to Z_G and all heights 5 meters or more above ground level to $Z_G + 5$ and effectively removing elevation features outside the height range of interest for dike detection. Then downsampling by factor 2 is performed to further decrease the computational cost per block and a Gaussian Filter is applied to reduce noise.

```
def dike_detection_udf(chunk):
    np_matrix = numpy.matrix(chunk)
    # limit height values to relevant range
    clipped = clip(np_matrix, Z_G, Z_G + 5)
    # Change resolution from 500x500 to 50x50
    sampled = downsample(clipped)
    # Apply a Gaussian Filter to smooth points
    smoothed = gauss_filter(sampled)
    # Detect edges
    edges = laplace_filter(smoothed)
    binary = edges < threshold
    # Close small gaps
    closed = binary_closing(binary)
    # Extract contours points
    polygons = find_contours(closed)
    return polygons
```

Listing 1 Pseudocode of the UDF outlining the operations for transforming point grid into potential dike segments

Once the transformed height map is smoothed, the Laplace filter and thresholding is applied resulting in a binary image of potential dike segments. However, the identification is not perfect. There are unconnected segments that belong to a single system and small segmented 'noise' objects appear in the image. In order to make connecting dike components as easy as possible in the next stage, we apply morphology operators based on mathematical set theory [21]. More precisely, *binary closing*, a sequence of dilation and erosion, is used to close small gaps between detected segments. Additionally, all segments that have no contact to the image borders are discarded, as dikes are expected to stretch over large areas comprising multiple blocks and even tiles.

While the resulting binary image identifies dike segments, it is not in a useful or space-efficient format, due to the fact that the image includes very large irrelevant areas around dike segments. Hence, the next step is to extract the dike segment contours as an ordered list of points, which are then transformed in the same way as described in Section 5.2 to obtain a GeoPandas dataframe containing all identified polygons.

5.4. Dike Connectivity Analysis

The potential dike segments obtained by the process described in the previous section rely solely on detection within a limited area of 5 by 5 kilometers. However, the Dutch dikes form a large connected system of closed dike rings and long dike lines along rivers. Therefore, our next step is to raise dike detection to a higher level. Based on the collected polygons from the previous stage, we create a geometric network [22] which is a graph structure consisting of nodes representing geometric objects in space, in our case individual polygons, and edges describing their spatial relationship. If two polygons intersect or touch each other, they are connected in the graph. This data structure allows us to identify connected components that most probably belong to a larger system of dikes, and to remove isolated segments.

Working on a tile level, the graph can be obtained by applying a *spatial join* to the GeoPandas dataframe with itself, which yields all pairs of intersecting polygons. Due to the filtering of irrelevant segments and closing of gaps between adjacent segments in the previous stage, the resulting number of collected polygons from all blocks is relatively small and thus GeoPandas' spatial dataframe queries are preferable to Spark's distributed operations. The resulting joined dataframe contains all required edge information to create the geometric network using Python's *networkx*. Finally, a connected component algorithm is applied to obtain sets of connected segments which are then combined into a single polygon object. Again, polygons that do not have contact with the tile border are discarded to further reduce the amount of false positives.

As a last step, the output from all individual tiles is combined by applying the described connectivity analysis once more, this time to the overall set of segments obtained from the tile-level analysis. This allows us to filter out more isolated segments that probably do not belong to any larger dike system, and merge adjacent segments into single components.

The final set of detected dike segments is written to GeoJSON.

5.5. Visualisation

The visualisation of the generated DEM and dike segments was created by overlaying our data on top of a map of The Netherlands. To this end we utilised the MapBox service [23].

Mapbox provides a powerful map API that allows its users to perform a wide range of tasks from simple data visualisation, as is our use case to more complicated navigation and live data interactions.

The visualisation procedure is straight forward, we initialised a Mapbox map, centred around The Netherlands and set the co-ordinate system to the Amersfoort co-ordinate system as this is how our dataset is indexed. Following this, the DEM data in image format was added to the maps source data using Mapbox.addSource() . The position of our DEM was encoded as bounding co-ordinates (xmin, ymin, xmax, ymax) into the file name. Once the source is loaded into the map object it can be easily overlayed using Mapbox.addLayer() . Dike overlay as well as overlay of flood plains follow this approach. In contrast to the DEM in PNG format, co-ordinate information for these polygon objects is encoded directly in the GeoJSON objects. We allow the user to freely combine the different layers by switching individual components on and off which facilitates the exploration of the Dutch topology as well as different flooding scenarios.

The polygon representations of both dikes and vector DEM plains facilitate a fast loading process of data as their spatial information is compressed into only few contour points. To allow for smooth interaction with the layered map we refrained from using our high resolution tile-based raster DEM but use an adjusted version in slightly lower resolution where tiles are merged into one single DEM.

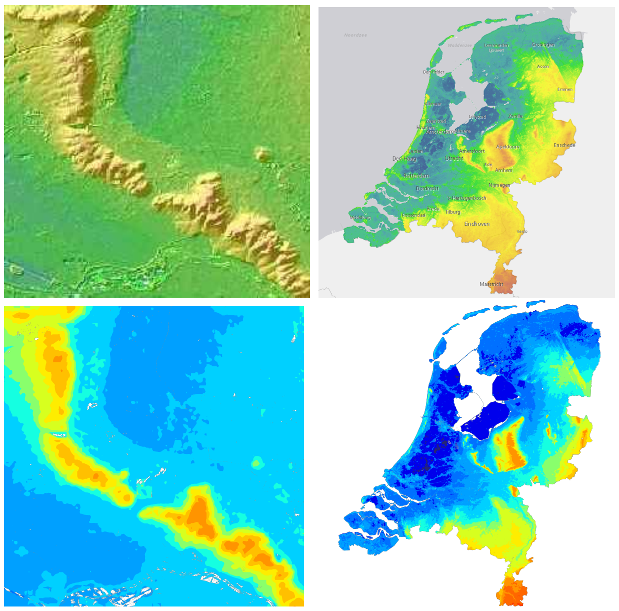
Figure 4 Reference elevation map (top) compared against our generated elevation map (bottom)

6. Evaluation and Experiments

Due to manifold technical obstacles arising during this project, the focus of our work lies on design and implementation of the data processing pipeline and generation of the targeted data product. Therefore, the extent of experiments performed on the final output is limited. With this in mind, this section will focus on validation of our results based on reference data. For both DEM and dikes we provide a side by side comparison of our output with the reference. Finally, preliminary experiments with our data products are presented and discussed.

6.1. DEM Evaluation

In order to evaluate the generated DEM we performed a visual comparison of the generated elevation images versus a reference elevation map of The Netherlands provided directly by AHN.

Figure 4 shows a comparison between the DEM generated by our pipeline (bottom) and the reference elevation map (top). From both comparisons, on tile-level and on country-level, we can see that overall quality of the generated DEM is pleasingly high. While our DEM's do not capture the same level of height transitions our intentions were to create a course grained elevation map and as such our goal was effectively accomplished.

6.2. Dike Detection Evaluation

To evaluate dike detection, reference dike data was acquired from the Dutch Rijkswaterstaat [24], a government organisation dealing with the design, construction and management of infrastructure facilities within The Netherlands. The reference data comes in form of a shape file and consists of connected line segments representing the dikes. In order to achieve an accurate comparison, the reference data was segmented into tiles according to the AHN2 tile structure.

First, a qualitative evaluation was performed. For the sake of brevity this procedure is based on taking a random sample of the generated tiles and comparing them with the reference tiles. Fig 5 shows what we believe to be adequate representations of the best (top), average (middle) and worst (bottom) case of true positive outputs from our detection algorithm. From the figure we can see that throughout, the difference is generally the amount of erroneously detected dikes that is introduced through irrelevant elevated features surrounding the dikes, which is understandable considering that our algorithm is based on height detection. In the worst case however, as well as seeing a lot of noise we do see loss of dike information. This could potentially be alleviated by decreasing the minimum height threshold for dike classification but would as a side effect introduce further false positives into an already noisy result.

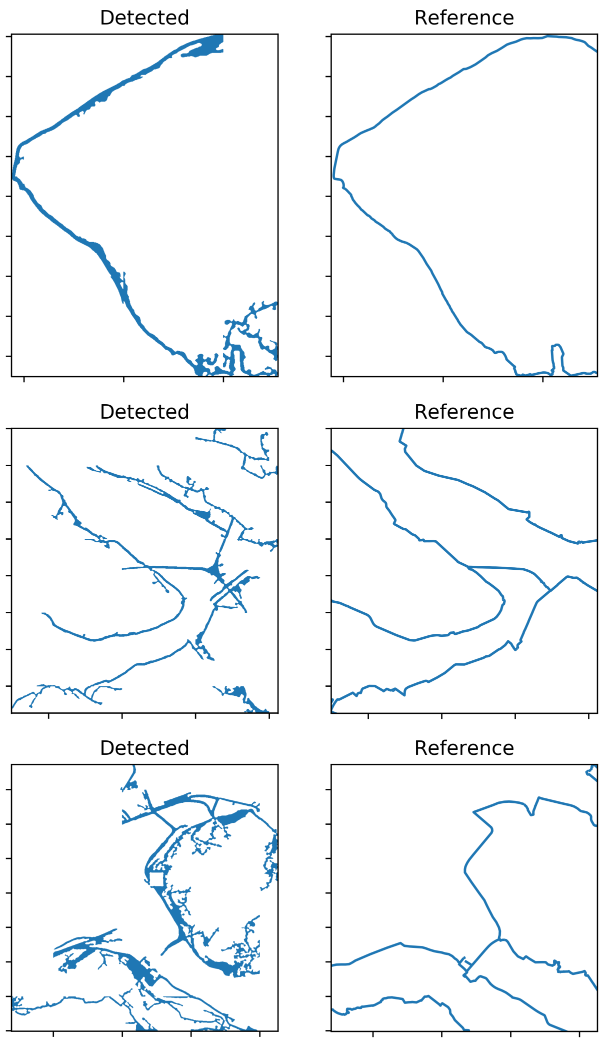

Figure 5 Best (top) Average(middle) and Worst(bottom) case outputs of our dike detection algorithm

Second, we provide a quantitative reliability assessment of our dike detection framework. More specifically, the number of detected dike segments overlapping with the reference dikes ($TP_{detected}$) as well as the number of reference dike segments that overlaps with our detected dikes (TP_{ref}) are determined and compared to the total counts ($N_{detected}$, N_{ref}) respectively. With this we define the following two measures of classification accuracy: *Precision* describes the fraction of detected segments that actually match true dikes, and *miss rate* gives us the fraction of reference segments that could not be detected by our classification method. Note that for these measures we consider individual segments before application of connectivity analysis. The equations and the actual results are presented in the following equations:

$$precision = \frac{TP_{detected}}{N_{detected}} = \frac{147}{206} = 71.36\% \quad (1)$$

$$miss\_rate = \frac{N_{ref} - TP_{ref}}{N_{ref}} = \frac{1051}{1845} = 56.96\% \quad (2)$$

In line with the visual examination, the precision of 71.36% underlines that the majority of our dike segments has correctly been classified, while there is still a significant amount of cases where other elevated features like bridges or roads are wrongly identified as dikes in areas where there are no dikes whatsoever. Apart from that, the high *miss rate* of 56.95% shows that our approach clearly suffers from missing a lot of true dike segments.

A visual overall comparison between the reference dike systems and our results supports the outcomes of the accuracy evaluation. Fig 6 shows that our dike detection framework is able to capture the rough layout of Dutch dike systems. Especially sea dikes along the west coast as well as in the north and around the Ijsselmeer could be detected with pleasing reliability. However, in line with the measured *miss rate*, fig 6 illustrates weaknesses of our approach as dike lines tend to be fragmented, with a significant amount of segments missing in between. Moreover, we see that most false positive detections occur in urban areas. Another significant part of false detections appear as straight lines at tile borders where smoothing and edge detection are obstructed by missing halo points from neighbouring tiles. While applying overall connectivity analysis reduces the number of false positives, it also results in loss of true dike segments due to fragmentation.

6.3. Experiments

In a preliminary experiment we examine which regions of the Netherlands would be flooded if there was no protection through dikes at all. Based on the generated vector DEM data, we identify all areas lying below sea level and present them in fig 7. Additionally, areas at height levels up to 3 meters above sea level are visualised in steps of 0.5 meters.

From the figure it becomes very clear that the Netherlands rely heavily on their dike systems. Without this protection all areas below sea level that are in some way connected to the sea would be flooded. This applies especially to the surroundings of the Ijsselmeer, comprising the provinces of North Holland, west Friesland and Flevoland, as well as South Holland. From our polygon model we estimate that 9213 km^2 would be affected by such a catastrophe. Interestingly, a rising sea level only leads to a moderate

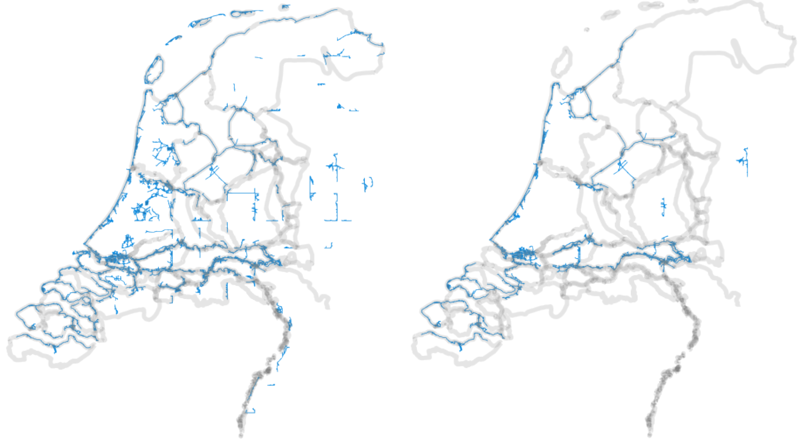
Figure 6 Overall comparison between reference dike data (grey) and our results (blue). Left: without overall connectivity analysis. Right: only connected components comprising at least 3 segments.

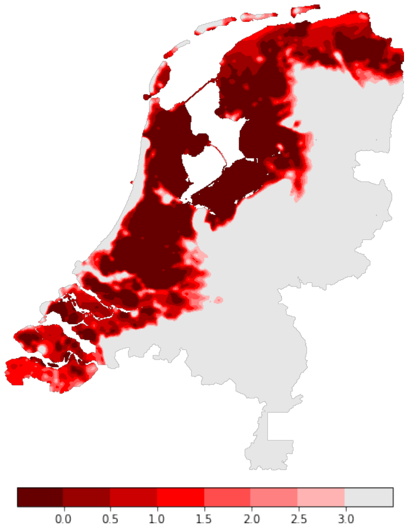

Figure 7 Critical areas that lie below (dark red) or up to 3 meters above sea level (lighter red).

increase in the flooded area. Our model predicts an area of 11981 km^2 being flooded if the sea level rises by 0.5 meter, 14773 km^2 for 1 meter.

7. Conclusions

To sum up, the project has been continuously driven by challenges posed by the massive size of the dataset. Many powerful tools could not be applied because they do not support the LAS/LAZ format and/or are not compatible with the Surfsara cluster configuration. However, we were able to overcome these challenges by developing a distributed processing pipeline that combines the potential of Apache Spark's resilient distributed datasets with the algorithmic strengths of standalone libraries like Python's *scikit-image* and *geopandas*.

Based on this technical approach, we successfully generated an image-based raster DEM of the entire Netherlands which provides coarse grained but reliable information about regions of equal height level, filtering out irrelevant non-ground features like buildings and vegetation. The same framework was applied to obtain additional vector-based information about these areas of equal height, which proved to be a space-efficient representation that can immediately be used for preliminary flood analysis.

From the analysis perspective, dike detection turned out to be the most challenging part of this research. The high resolution of the raw data including lots of irrelevant information together with its unfeasible volume made it difficult to appropriately reduce the data to a workable amount that still includes the small-scale dike details needed for automatic classification. Our edge detection approach was able to reliably detect dominant sea dikes where the surrounding terrain is relatively homogeneous. However, smaller river dikes in more urban areas remain a challenge due to distorting features such as elevated roads, bridges and large constructions. In these cases, finding appropriate parameter values for Gaussian smoothing and thresholding is non-trivial and needs further optimisation.

Flooding analysis had to be limited to preliminary flood plain prediction as more sophisticated simulations based on dike breaches would require not only highly accurate detection of fully connected dike systems but also deeper studying of flood dynamics.

Considering the discussed challenges, this project has great potential for further research. Future work might focus on three different aspects. First, special effort should be made to build tools for distributed point-cloud processing, and implementation should be moved to a up-to-date cluster architecture which facilitates full functionality of geospatial Spark libraries like *Geotrellis* and *GeoSpark*. Second, more time should be spent on optimising automatic detection of dike systems. Intelligent parameter tuning, incorporation of machine learning based on local point features or deep learning appraoches are conceivable. And third, with improved outcomes based on the first two aspects, it is desirable to apply true flood simulations to the geospatial models, including water flow modelling, susceptablity of dikes and more. Insights from such studies are expected to be vital for the Dutch and the existence of their country, as improvements in flood defense rely more and more on high-throughput computational simulations.

8. References

[1] M. Zaharia, M. Chowdhury, M.J. Franklin, S. Shenker, and I. Stoica. Spark: Cluster computing with working sets. *HotCloud*, 10(10-10):95, 2010.

[2] J. Van Alphen. The delta programme in the netherlands: a long-term perspective on flood risk management. In *Floods: from risk to opportunity. Proceedings of the 5th International Conference on Flood Management, Tokyo, Japan*, pages 27–29, 2011.

[3] H. Most and M. Wehrung. Project floris - flood risks and safety in the netherlands : flooding in the netherlands - probabilities and consequences, 2003.

[4] H. Most and M. Wehrung. Dealing with uncertainty in flood risk assessment of dike rings in the netherlands. *Natural Hazards*, 36:191–206, 2005.

[5] G. Priestnall, J. Jaafar, and A. Duncan. Extracting urban features from lidar-derived digital surface models. *Computers, Environment and Urban Systems*, 24:65–78, 2000.

[6] T.G. Thomas and J.J.R. Williams. Large eddy simulation of turbulent flow in an asymmetric compound open channel. *Journal of Hydraulic Research*, 33:27–41, 1995.

[7] X. Meng, N. Currit, and K. Zhao. Ground filtering algorithms for airborne lidar data: A review of critical issues. *Remote Sensing*, 2(3):833–860, 2010.

[8] N. Pfeifer, A. Kostli, and K. Kraus. Interpolation and filtering of laser scanner data - implementation and first results. *International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences*, 32 (part 3/2):153–159, 1998.

[9] G. Vosselman. Slope based filtering of laser altimetry data. *International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences*, 33 (part B3/2):935–934, 2005.

[10] J. Kilian, N. Haala, and M. Englich. Capture and evaluation of airborne laser scanner data. *International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences*, 31 (part B3):383–388, 1996.

[11] E.S. Anderson, J.A. Thompson, and R.E. Austin. Lidar density and linear interpolator effects on elevation estimates. *International Journal of Remote Sensing*, 26(18):3889–3900, 2005.

[12] Y. *et al.* Bao. Classification of lidar point cloud and generation of dtm from lidar height and intensity data in forested area. *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Science*, 37 B3/B:313 –318, 2008.

[13] K. Liu and J. Boehm. Classification of big point cloud data using cloud computing. *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, XL-3/W3:553–557, 2015.

[14] J. Boehm, K. Liu, and C. Alis. Side loading – ingestion of large point clouds into the apache spark big data engine. *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, XLI-B2:343–348, 2016.

[15] C. Wang, F. Hu, D. Sha, and X. Han. Efficient lidar point cloud data managing and processing in a hadoop-based distributed framework. *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, IV-4/W2:121–124, 2017.

[16] ESPG.IO. Coordinate systems worldwide. coordinate reference systems for "netherlands". `https://epsg.io/?q=Netherlands`.

[17] Las specification version 1.2. Technical report, ASPRS, 2008.

[18] Lastools. `https://rapidlasso.com/LAStools/`, Oct 2018.

[19] R.C. Gonzalez and R.E. Woods. *Digital Image Processing*. Prentice Hall, 2 edition, 2002.

[20] L. Van Nieuwenhuijze. The challenge of flood risk management, 2015.

[21] J. Serra. *Image Analysis and Mathematical Morphology.* Academic, 1982.

[22] Marc Barthélemy. Spatial networks. *Physics Reports*, 499(1-3):1–101, 2011.

[23] Mapbox. =https://www.mapbox.com/.

[24] Rijkswaterstaat. Home. `https://www.rijkswaterstaat.nl/`, Oct 2018.

Appendix A Work Distribution

Table 1 Who worked an which parts of this project?

Preprocessing	Maki Gradecak
DEM Generation	Fiona Lippert
Dike Detection	Fiona Lippert
Visualization	Maki Gradecak and Peter Petkanic
Report	Maki Gradecak and Fiona Lippert