Flow on imprecise terrains

Theunissen, M.L.M.

Award date:
2013

Technische Universiteit
Eindhoven
University of Technology

Department of Mathematics and Computer Science
Den Dolech 2, 5612 AZ Eindhoven
P.O. Box 513, 5600 MB Eindhoven
The Netherlands
www.tue.nl

Flow on Imprecise Terrains

Master's Thesis

Marlous Theunissen

Supervisor
dr. Herman Haverkort (TU/e)

Section
Algorithms and Visualization

Date
April 9, 2013

Acknowledgements

After almost eight years of studying at Eindhoven University of Technology (TU/e), I proudly present my Master's thesis. This graduation project would not have been the same without the support of many people. I wish to express my gratitude to my supervisor, Herman Haverkort, without whom this text could not have been written yet, since he did a great job in tutoring me during this graduation project. Thank you for your time, your lovely stories about Emma and most of all, thank you for the project discussions, which lead to some fruitful ideas. I would also like to thank Anne, who served as a second tutor for me when I had an urgent matter and Herman could not help. I would like to thank Sander, Q and Alex for the in-depth discussions about algorithmic problems, and the fellow algorithms graduates for the tea breaks once in a while. Furthermore, I should not forget 'the guy from the hallway', whom I now know as Roeland. Although he and others tried to convince me of doing a PhD afterwards, I did not listen.

I also want to thank my parents, for their love, support, but most of all their patience when I again was talking about the interesting algorithmic details of a problem. Another person with a lot of patience is Erik, who has had a lot of chatter to endure but who always kept listening – or at least, pretended that he did. He is a wonderful guy, and was open to discussions about my project whenever I needed that and helped me when I got lost in the great world of Mathematics. Also many love and thanks to Bieflap and Stooflap, the hamsters which made me smile again after a proof went wrong or after a full day of debugging.

Finally, I want to thank all the students with whom I enjoyed classes or who participated in my social life – you too Jeroen. You all did a great job in helping me with the successful completion of this project.

Abstract

Driemel et al. [DHLS11] introduced a new type of algorithms for flow computations on terrains using the Single Flow Direction (SFD, steepest-neighbour) method. These algorithms can deal with terrains represented by a graph $G = (V, E)$, with a certain elevation range $[low(v), high(v)]$ for every vertex $v \in V$. The elevation range can be seen as an uncertainty in the elevation of the vertex; the elevation-bounded model is therefore an instance of an uncertainty model. An alternative method to model the uncertainty in a terrain is by defining bounds on the elevation difference between adjacent vertices: the slope-bounded model. We investigate both uncertainty models and gain insight in the behaviour in practice of the algorithms by Driemel et al. and our extension to the Multiple Flow Direction method (MFD, where water flows to all neighbours with a lower or equal elevation).

We prove that it is NP-hard to determine whether water can flow from one point to another point in the slope-bounded SFD model. For the remainder of the project, we therefore concentrate on the watershed algorithms by Driemel et al. for elevation-bounded models. By interpolating neighbourhoods of different sizes, we obtain several elevation-bounded model instances, based on a Digital Elevation Model (DEM, in which vertices have a fixed elevation). However, this still does not capture the actual behaviour of water flow. Additionally, we therefore apply a flooding technique (lifting the elevations of each vertex until it is able to leave the terrain) to the data sets. By applying both techniques, the watershed algorithms return plausible results.

Driemel et al. define a potential watershed as the region from which water may flow to a certain set of vertices Q, and a persistent watershed as the region from which water can only escape by flowing through Q. We define the certainty of the watersheds of Q as the ratio of the persistent and potential watershed. The algorithms frequently show high uncertainty when applied to small blocks of the terrain. Lower uncertainty is observed for blocks intersecting a river bed. The reliability of the results is therefore still questionable. The algorithms still lack robustness; a small difference in the input can seriously affect the resulting watersheds. We therefore conclude that current results can only be interpreted by users who know the area and algorithms very well. Either further assistance is required for selecting the area of interest, or the algorithms have to be improved.

Contents

CHAPTER 1

Introduction

In Geographic Information Science (GIS), a lot of research has been done on the behaviour of water on a terrain. With a realistic simulation of, for example, the water flow on a terrain, one can analyse certain effects and risks of, for instance, flooding, or give a better overview of the water flow after a heavy downpour. However, since we do not have the exact terrain properties available of every square meter of the Earth's surface at any moment in time, elevation data of this surface is used to create elevation models to give an approximation of the area. This elevation data can be obtained as described by Haverkort and Toma [HTar] by using methods such as extracting the data from the contour lines on a topographic map, or by using a more recent technique as LIDAR (Light Detection And Ranging) or SONAR (SOund Navigation And Ranging). Two disadvantages of these type of measurements are that they often contain noise and they give us a discrete representation of the surface. After all, the elevations of points for which no measurements are done need to be estimated. Therefore, calculations on this elevation data can never yield a fully realistic simulation with complete certainty. In this report we discuss this discrete representation of a terrain, the *terrain models*, and will refer to these models as terrains instead of the actual surface that is represented by the data.

Traditional preprocessing
Most of the developed algorithms that are currently in use in common GIS-applications base their calculations on a digital elevation model (DEM). This DEM consists of a set of points, where each point has an exact elevation. The fact that elevation data of a terrain contains noise, due to the measurements, and the fact that the estimation of an elevation value at a specific point is done by, for instance, interpolation, both give rise to an uncertainty interval in which the elevation of the point in the real terrain probably would lie. That means that in order to use these algorithms, when extracting elevation data from e.g. topographic maps or LIDAR point clouds, we need to select one of the many plausible elevation values for each specific point. We call this a realization of the terrain – a selection of elevation values for the points of the terrain. By choosing such a value for each point, a lot of other possible realizations of the terrain are discarded. However, due to several errors, it is not sure whether the selected value is the actual value. As a result, preprocessing of a DEM is needed before doing actual calculations. As described by Cheng-Zhi and Lijun [CZL12], doing flow accumulation calculations in real applications requires a DEM preprocessing algorithm to remove depressions and flat areas in advance. Preprocessing techniques that are commonly used involve filling the closed depressions (flooding) and detecting drainage routes in the input data; removing local extrema; or stream burning, a technique that overlays the data with a vector input of the river network to improve

the elevation values of data points along rivers [dPCRM08]. Image processing techniques such as smoothing are used to remove small-scale roughness caused by interpolation techniques.

Consequences

Each preprocessing technique influences the quality of the calculations on the resulting DEM. Two channel mapping techniques, techniques for extracting drainage networks from DEMs, are compared by Lindsay [Lin06]. The channel-initiation algorithms simulate overland flows to locate channel heads, where the valley-recognition algorithms identify channel grid cells based on specific topographic profiles, such as a 'v'-shape. Both approaches extract a drainage network from the input model. However, if the error magnitude of a DEM is moderate to large, and/or if the degree of spatial autocorrelation in the error surface is low, Lindsay states that the channel network extracted will likely contain significant artefacts due to errors. Furthermore, the elevation errors influence the positions of the channel heads. Lindsay also states that 'algorithms are also hindered by the surface depressions occurring on rough surfaces, which is associated with a more detailed representation'. Note that a gridded DEM suffers from the problem that elevations are sampled on a regular grid, which may under-sample the terrain in rugged areas and oversample it in smooth areas. Wise [Wis07] uses six different DEM creation methods on the same area, a small catchment in Devon (UK), to investigate the effects on the results of several DEM algorithms. As a result, predictions differed in some cases over 200% for surface run-off and a difference of 25% was measured for hourly flow values. The first result was instigated by the presence of interpolation artefacts in the DEM, leading to completely unrealistic predictions. The overall conclusion of Wise supports the idea that DEM artefacts have little effect on broad-scale results, but that they can have important consequences for local-scale results, which is the scale at which most DEM analysis algorithms work. Errors in DEMs are usually gross errors or blunders, systematic errors due to bias in the data and random errors [FT06]. However, not only the errors in DEMs itself causes errors in the algorithms results. Also the selected flow models (Section 2.3) have influence on the results, as reported by Zhou and Liu [ZL02].

Sources of uncertainty

As we can see, there are several sources of error that appear when computing with terrain models. We can divide the errors, influencing the results of computations, roughly into four categories. First of all, the measurements conducted to retrieve elevation values for a number of points are not precise; we call these errors *measurement errors*. The equipment used for the measurements is ultimately error-prone and often returns a height with a known error bound or a height interval instead of a fixed value [GKLS12]. Second, we have the errors caused by creating a terrain model from the measured data. We will call these errors *interpolation errors*. As a third, the sampling rate has an effect on the results of the elevation models, denoted by *sampling rate errors*. With more data points a higher resolution model can be created and more details of the terrain are captured. Last but not least, we have the *interpretation errors*. These errors are either caused by the inherent shortcomings of an algorithm or by the inability to effectively capture certain area-specific properties in the terrain model. For instance, when a valley has a bridge crossing a river at a height that is much higher than the actual height of the river, the elevation models possibly capture this as a dam, looking at the elevation values, whereas in practice, the water could actually flow underneath the bridge.

Taking uncertainty into account

Due to the errors described above, one may want to keep some uncertainties that can be extracted from the data set available in one way or another, when doing calculations on the terrain. We call a model that takes uncertainty into account an *uncertainty model*. One way to translate these uncertainties into a terrain model is described by Driemel et al. [DHLS11], where each point in the input has an elevation range $[e_l, e_h]$ instead of a fixed elevation. That is, a point p has a minimum elevation

and a maximum elevation and the exact elevation is not known. In the rest of this document, this model is denoted as the *elevation-bounded model*. Using such a terrain model that keeps uncertainties into account, we want to have a notion of uncertainty in the output of the algorithms. Driemel et al. described several algorithms to translate this uncertainty in the input to the output, beginning with algorithms that calculate uncertainty ranges for the watersheds for point sets.

Water flow
Algorithms simulating water flow base their calculations on a so called *flow model*. It is very difficult to describe the exact behaviour of water on a terrain when the terrain is only represented by a grid of points with elevations. Every algorithm that simulates water flow is a compromise between all possible flows, since a grid representation simply limits the possibilities of the real flow. Water has a typical behaviour of flowing downwards. To approach this flow behaviour, several flow models have been described to determine the flow direction and the flow amount of water leaving a certain point [GP08]. Driemel et al. base their algorithms on the flow model where water flows to the steepest descent neighbour.

This report
In this report, we extend the watershed algorithms as introduced by Driemel et al. for a second flow model, where water can flow to multiple neighbours. We look at two different uncertainty models. The first model, the elevation-bounded model, is shortly described in Section 2.2.1. Based on an equidistant grid model, where all edges between points have the same distance, we show the nesting properties of watersheds in this model. The second uncertainty model we will look at, is the *slope-bounded model*, which is introduced in Section 2.2.2. For this second model, we show that determining whether water could flow from a certain point to another point is NP-hard. Afterwards, we switch back to the first uncertainty model and take a closer look at the behaviour in practice of the watershed algorithms by using data from the Neuse River Basin in North Carolina. We describe several ways to select the elevation ranges such that the uncertainty is taken into account and discuss the results.

CHAPTER 2

Preliminaries

In this chapter we will introduce the relevant models and terms that are used in the rest of the report. We will first define the terrain models used, describing a terrain by a set of points in the xy-plane with for each point an elevation. Next, the two uncertainty models studied are explained. Since the report is about flow on imprecise terrains, we also need to define which flow models we take into account. This is done in Section 2.3. Last but not least, we conclude this chapter with a brief overview of the relevant algorithms introduced by Driemel et al. [DHLS11] that are used to obtain our results in Chapters 4, 7 and 8.

2.1 Modelling terrains

A digital elevation model used in GIS applications could be described as a grid with cells, for which every cell (or every centre of the cell) is assigned an elevation. Such a model can be modelled as a graph $G = (V, E)$, in which each point $v \in V$ represents the centre of a grid cell and has a property 'elevation', and each edge $e \in E$ represents the adjacency of two adjacent (neighbouring) cells in the grid. The neighbourhood of a point set P, denoted by $N(P)$, is the set of vertices that have an edge to a vertex in P and are not part of P itself. $N(P) = \{s : s \notin P \wedge \exists t \in P : (s,t) \in E\}$.

2.1.1 Irregular network

An irregular network is a network of irregularly distributed vertices and edges with three-dimensional coordinates (x, y, z), defining the location and elevation value of a vertex. To be able to reduce the required storage space, it is possible to allow a lower sampling density on parts of the terrain that are less interesting. To combine different sampling densities within a graph, one could use an irregular network in which each vertex drains to one or more of its neighbours. We refer to this irregular network as the *network model*, and unless indicated otherwise we assume that, from every vertex, water flows down along the steepest incident edge.

2.1.2 Regular square grid

A regular terrain is a terrain for which there exists a periodic structure in the input data (a graph) of the terrain. A typical example is a regular square grid; for a square of size $\delta \times \delta$, the coordinates of the vertex in the i'th column and j'th row can be defined as $(i \cdot \delta, j \cdot \delta)$.

The digital elevation models used in most GIS computations are regular square grids. These grids can be seen as a matrix of unit size squares, where each square has a given elevation. There are two types of these grid models: one where the cells of the grid are modelled as squares that have a certain elevation and one where the corners of the square grid cells are used as the basic elements of computations. We will use the type for which the centres are modelled as vertices in a graph, and we count the rows from top to bottom.

For each square, say of size $\delta \times \delta$, the neighbours are given by the eight surrounding squares. However, the distances (measured using the centre points of the squares) from such a square to its neighbours are not all equal. For example, the distance to a north-east neighbour is $\sqrt{2}\delta$, which is larger than the distance δ to its north neighbour (see Figure 2.1).

2.1.3 Regular triangular grid (hexagonal grid model)

A different regular grid is a grid where hexagons are repeated. Connecting the centres of the hexagons for all neighbouring cells yields a regular triangular grid. In this regular triangular grid the lengths of the edges are all equal. This property simplifies several flow computations; since steepest neighbour tests are based on the proportion of the distance and the accompanied elevation difference, one only need to look at the elevation differences when all edges have the same length. Therefore we will focus at computations with equidistant grids where each distance between two neighbours is equal. In particular, we choose this regular triangular grid. In this report, we will also denote this regular triangular grid as the **hexagonal grid model**, because the vertices define the centres of the cells of a regular hexagonal grid, as shown in Figure 2.2. In this way, we can talk about cells as is done for GIS applications. An example of a hexagonal grid with elevations on each cell, is given in Figure 2.3.

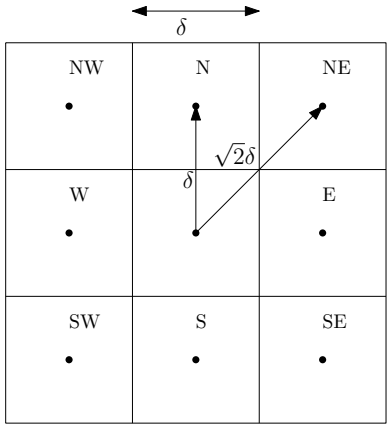


Figure 2.1: Unit grid DEM

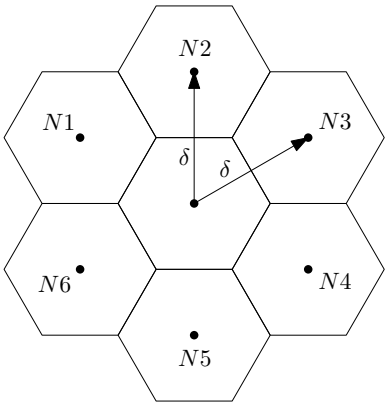


Figure 2.2: Hexagonal grid

2.2 Modelling uncertainty

2.2.1 Elevation-bounded model

As briefly described in the introduction, one of the two uncertainty models we will take a look at, is the *elevation-bounded model*, as described by Driemel et al. [DHLS11]. In this model, each point in the input has an elevation range $[low(v), high(v)]$ instead of a fixed elevation. That is, a point p has a minimum elevation $low(v)$ and a maximum elevation $high(v)$ and the exact elevation is not known. In Figure 2.4, an example is given of an elevation-bounded model. Note that in this type of model, all elevations of the different points can be selected independently of each other. (In the second model, described in Section 2.2.2, this is not the case.)

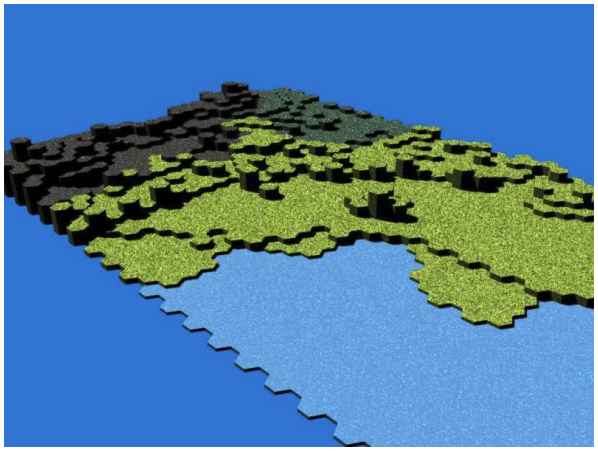


Figure 2.3: Example of elevations embedded in the hexagonal grid.
Image taken from http://geekanddad.wordpress.com/category/games/.

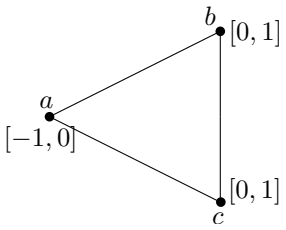


Figure 2.4: Example of an elevation-bounded model in an equidistant grid. The elevation of each vertex can be selected independently.

We use the definition of an imprecise terrain $\mathcal{T}$ as presented by Driemel et al. An imprecise terrain is defined as a graph $G = (V, E)$, where each vertex $v \in V$ has a third property, which represents its imprecise elevation. The bounds of this elevation are denoted by $low(v)$ and $high(v)$. A realization R of $\mathcal{T}$ consists of graph G together with an assignment of elevations to the vertices V_R, where the assigned elevation $elev_R(v)$ is at least $low(v)$ and at most $high(v)$. The edge set E_R is the set of edges E mapped to the vertices with the assigned elevations. We will use the notation of R^- for the realization where all vertices are assigned the lowest elevation; i.e. such that $elev_{R^-}(v) = low(v)$ for every vertex v. Similarly, R^+ represents the realization such that for every vertex v it holds that $elev_{R^+}(v) = high(v)$. The set of all realizations of an imprecise terrain $\mathcal{T}$ is denoted with $\mathcal{R}_\mathcal{T}$.

Note that Driemel et al. [DHLS11] base their definitions and proofs on the Single Flow Direction model, as explained in Subsection 2.3. Further on in this report, we will extend this approach with the Multiple Flow Direction model.

2.2.2 Slope-bounded model

Next to the elevation-bounded model, we will discuss the *slope-bounded model*. The slope of two points is defined here as the ratio of the vertical distance divided by the horizontal distance; for a model where all edges have the same distance, the slope is simply denoted by the elevation difference between the two points.

As opposed to the first model where points have elevation bounds, in the slope-bounded model, or *slope model*, the edges are given bounds on their slopes – the *slope range* $[low(e), high(e)]$. In this model, the elevation of a vertex is relative to the elevation of its neighbour, and we can only retrieve information about the difference in elevation between two vertices instead of exact elevation details. The slope range $slopeRange(e)$ of an edge $e \in E$, $e = (v_1, v_2)$, gives a bound on the possibil-

ities of the elevation difference between vertices v_1 and v_2, the slope $slope(e)$. For example, for an edge with length one, we may have $slopeRange(e) = [0,2]$ and $slope(e) \in [0,2]$. In that case, assuming that the elevation of v_1 is denoted by $elv(v_1)$ we know that v_2 has a possible elevation range of $[elv(v_1), elv(v_1)+2]$.

Also for this model, we give a definition of an imprecise terrain $\mathcal{T}$. An imprecise terrain is defined as a graph $G = (V, E)$, where each edge $e \in E$ has a third property, which represents its imprecise slope. The bounds of this slope are denoted by $low(e)$ and $high(e)$. A realization R of $\mathcal{T}$ consists of graph G with edges E_R with an assignment of slopes to the edges, where the assigned slope $slope_R(e)$ is at least $low(e)$ and at most $high(e)$ and there exists an assignment of elevation values to the vertices in V such that all selected slopes indeed obey the slope ranges. The set V_R is the set of vertices E mapped to the edges with the assigned slopes. The set of all realizations of an imprecise terrain $\mathcal{T}$ is denoted with $\mathcal{R}$.

Note that the slope of an edge is directed, that is, $slope((u,v)) = -slope((v,u))$ and the elevation of a vertex can be described by $elv(v) = elv(u) + slope((u,v))$ where $slope((u,v)) \in [low(v), high(v)]$. As a result, after choosing a slope for a certain edge, the *valid slope range*s (the range for which there still exists a realization of the model) of other edges in the graph may be effected, as in the example in Figure 2.5. When selecting a slope 0 for edge (a,b), the slope ranges of the other two edges stay the same as depicted in the Figure. However, when selecting a slope 1 for this edge, the only possibility for the remaining edges, is to select a slope of 0 for edge (b,c) and a slope of -1 for edge (c,a), otherwise a realization does not exist.

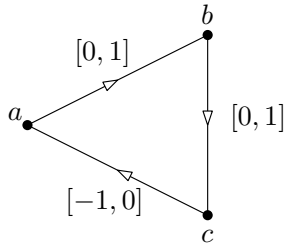


Figure 2.5: Example of a slope-bounded model in an equidistant grid. Selecting a slope for one edge, influences the possible slope values for other edges.

2.3 Modelling flow

When modelling the flow of water, one could choose several types of flow algorithms to determine the flow direction and flow amount of water leaving a certain point [GP08]. We choose either the Single Flow Direction (SFD), the so-called *D8* method, or the Multiple Flow Direction (MFD) model. In the first model, the water always flows to the steepest descent neighbour of that point. In the second model, the water is divided over a subset of the neighbours that have a lower elevation than the point itself. However, there exist a large number of definitions of this division. The most commonly used methods are the *FD*8 method and the *D*-∞ method. In the first method, water flows to all lower neighbours with a percentage based on the slope of each neighbour. The second algorithm only allows water to flow to at most two neighbours, but here, too, the amount of water is based on the slope of the neighbour points. In the calculations of the watershed algorithms as described in the next section, it is also possible that water flows to a neighbouring point with the same elevation as the current point. In the case of the SFD method this is only possible when there are no other points with a lower elevation. In the case of MFD, it depends on the exact definition. The MFD method as we apply it in Section 4.2 also allows water to flow to a neighbour with the same elevation as the point where water flows from.

If water from p reaches a node $q \in V_R$ then we write $p \xrightarrow[R]{} q$ ("p flows to q in R"), and for technical reasons we define $p \xrightarrow[R]{} p$ for all p and R. We use π to describe a **flow path** in the graph G of an imprecise terrain $\mathcal{T}$. That is a path with no repeated vertices, which carries water to a local minimum in a realization R and visits all vertices of this local minimum in that specific realization. Given a path π and any pair of points p,q in π, the order of p and q in π is determined by the order in which water flows from one point to another; $p \xrightarrow[\pi]{} q$ if $p \xrightarrow[R]{} q$. For any pair of nodes p, q in π, we also write $p \xrightarrow[\pi]{} q$ if $p \xrightarrow[R]{} q$, that is, π contains p and q in this order. We denote with $\pi[p,q]$ the subpath of π from p to q, including these two nodes. For a set S of realizations ($S \subseteq \mathcal{R}_\mathcal{T}$), $\Pi(S)$ represents the set of all flow paths in any realization in S.

2.4 Watersheds

In a realization R, where for each node an elevation value is assigned, the **watershed** of a node q is defined as the union of nodes that flow to q in R. We can also write this as $\mathcal{W}_R(q) := \{p : p \xrightarrow[R]{} q\}$. The discrete watershed of a set of nodes Q is now defined as $\mathcal{W}_R(Q) := \bigcup_{q \in Q} \mathcal{W}_R(q)$.

Unfortunately, given an imprecise terrain $\mathcal{T}$, we can not simply calculate the discrete watershed of a node (or set of nodes) due to the fact that the elevation values are now ranges of values instead of one single value. Therefore, Driemel et al. introduced the notion of potential watersheds, Q-avoiding potential watersheds, core watersheds and persistent watersheds. We will give a brief introduction to these four types of watersheds here, for the complete explanation we refer to Driemel et al. [DHLS11].

The **potential watershed** of a set of nodes Q in a terrain $\mathcal{T}$ is defined as

$$\mathcal{W}_\cup(Q) = \bigcup_{R \in \mathcal{R}_\mathcal{T}} \mathcal{W}_R(Q),$$

which is the set of nodes such that there exists a flow path in the set of all possible realizations such that water leaving p will eventually also enter node $q \in Q$ on that flow path. The easiest way to think of this potential watershed is by taking the union of the nodes on possible flow paths to Q for the different realizations, which results in the definition given here. We will write $\mathcal{W}_\cup(q)$ to denote the potential watershed of a single node q, slightly abusing the notation. Besides the union of all possible watersheds of a set Q for every realization R, we can describe the potential watershed by looking at the possible flow paths and the nodes contained in them:

$$\mathcal{W}_\cup(Q) = \{p : \exists \pi \in \Pi(\mathcal{R}_\mathcal{T}), \pi \ni p \ \exists\ q \in Q : p \xrightarrow[\pi]{} q\}.$$

The potential watershed of Q is also called the *maximal* watershed, since it represents the largest set of nodes for which water could possibly flow to a node in Q in any of the realizations in $\mathcal{R}$ [DHLS11]. The analogous definition to this yields the set of nodes p from which water flows to Q via *any* induced flow path that contains p, and is given by

$$\mathcal{W}_\cap(Q) = \{p : \forall \pi \in \Pi(\mathcal{R}_\mathcal{T}), \pi \ni p \ \exists\ q \in Q : p \xrightarrow[\pi]{} q\}.$$

We call this the **core watershed** of Q. However, as explained by Driemel et al., this definition is a bit too restrictive. In the case of overlapping elevation ranges of neighbouring nodes, each node could become a local minimum in some realization, so that water from that node will not reach Q, while it is clear that water from these nodes will eventually end up in a node of Q. A definition that takes these situations into account, is the definition of the Q-**avoiding potential watershed** of a set nodes S:

$$\mathcal{W}_\cup^{\setminus Q}(S) = \bigcup_{\pi \in \Pi(\mathcal{R}_\mathcal{T})} \bigcup_{s \in S} \{p : (p \xrightarrow[\pi]{} s) \wedge (\pi[p,s] \cap Q = \varnothing\},$$

which represents the set of nodes that have a potential flow path to a node $s \in S$ that does not pass through a node of Q before reaching s. Nodes that flow to a node in Q in at least one realization and which get stuck in a local minimum inside the potential watershed of Q in all other realizations, will therefore not appear in the Q-avoiding potential watershed of S, if S lies outside $\mathcal{W}_{\bigcup}(Q)$. So the nodes that do appear in Q-avoiding potential watershed of S are the nodes that are able to leave the potential watershed in some realization and can thus escape from flowing to Q.

Last, but not least, we define the **persistent watershed** of set nodes Q, also referred to as the minimal watershed, which is a subset of the nodes in the potential watershed of Q. In words, a persistent watershed of Q is the set of nodes that occur within the potential watershed of the set Q, for which the only escape from flowing to a node in Q is ending up in a local minimum within this potential watershed for any possible realization. Or, as Driemel et al. describe it, the set of nodes that cannot have a high enough elevation such that the water from those nodes could escape from the potential watershed of Q. The formal definition can be written as

$$\mathcal{W}_{\bigcap}(Q) := \left(\mathcal{W}_{\bigcup}^{\setminus Q}((\mathcal{W}_{\bigcup}(Q))^c)\right)^c ,$$

and it can be shown that $\mathcal{W}_{\bigcap}(Q) \subseteq \mathcal{W}_{\bigcup}(Q)$. In this report, we only use the definition of a Q-avoiding potential watershed of a set S to find the persistent watershed of Q. That means that we assume that every set S when writing about the Q-avoiding potential watershed of S, does not contain any point from the potential watershed of Q. Furthermore, slightly abusing the notation, the shorthand notation 'Q-avoiding potential watershed' refers to the Q-avoiding potential watershed of the complement of the potential watershed of Q.

Given these definitions, the authors of [DHLS11] also propose an algorithm for calculating the potential watershed of a set of nodes, the POTENTIALWS(Q) algorithm.

Algorithm POTENTIALWS(Q)
1. For all $q \in Q$: Enqueue (q, z) with key $z = low(q)$
2. **while** the Queue is not empty
3. **do** (q', z') = DequeueMin()
4. **if** q' is not already in the output set
5. **then** Output q' with elevation z'
6. Enqueue each $(p, z) \in$ EXPAND(q', z')

The EXPAND(q', z') algorithm calculates for each neighbour p of q' whether there exists a realization R such that water flows from p to q', where q' has elevation z' or higher in R, and returns p together with the minimum elevation for which such a realization R exists. We will use the steps in this algorithm later on in this report to simplify the algorithm in the case of equidistant graphs, where the length of all edges are equal, and we will extend the algorithm for the MFD situation. However, for the details of this EXPAND(q', z') algorithm, we refer to [DHLS11].

For completeness, we also give the pseudocode of the algorithms that calculate the Q-avoiding potential watershed of a set S, Q-AVOIDINGPOTWS(S, Q), and the persistent watershed of a set Q, PERSISTENTWS(Q), as described above.

Algorithm Q-AVOIDINGPOTWS(S, Q)
1. For all $s \in S$: Enqueue (s, z) with key $z = low(s)$
2. **while** the Queue is not empty
3. **do** (s', z') = DequeueMin()
4. **if** s' is not already in the output set $\wedge s' \notin Q$
5. **then** Output s' with elevation z'
6. Enqueue each $(p, z) \in$ EXPAND(s', z')

Algorithm PERSISTENTWS(Q)
1. $\mathcal{P} \leftarrow$ POTENTIALWS(Q)
2. $\mathcal{A} \leftarrow$ Q-AVOIDINGPOTWS$(\mathcal{P}^c, Q)$
3. Output $\mathcal{A}^c$

2.5 Expanding a node in an equidistant grid

For the SFD method, the water flow depends on which point is the steepest descent neighbour. Driemel et al. [DHLS11] uses a slope diagram for the EXPAND algorithm to determine in which cases a point can be the steepest descent neighbour. For the exact definition of the slope diagram as we use it, we refer to the definition as defined by Driemel et al. Each neighbour point q_i of p has a distance δ_i to p in the (x, y)-projection and a highest possible elevation value $high(q_i)$. These two values are used as the coordinates for the point $\hat{q}_i$ in the slope diagram. We are interested in the subset of neighbours $q_1, q_2, \ldots$ that appear in counter-clockwise order along the boundary of the convex hull of the slope diagram, starting from the left most point and continuing to the lowest point. The neighbours that do not lie on this lower left chain are ignored, since these points have no potential for being the steepest descent neighbour. The halfplane in the slope diagram that lies above the line through $\hat{q}_i$ and $\hat{q}_{i+1}$ is denoted by H_i. $U(p)$ is the intersection of the halfplanes $H_1, H_2, \ldots$, the halfplane right of the vertical line through the left most point and the halfplane above the horizontal line through the bottommost point of the convex chain.

One important observation in the case of an equidistant model, is that a slope diagram of a point p, called for in the EXPAND function in line 6, can be described by a single neighbour point. To be more precise, exactly that neighbour point with the minimal maximal elevation of all neighbour points of p. The slope diagram of a point p exists of all neighbours $q \in N(p)$, where each q is placed in the diagram at coordinate (x, y), where x equals the distance between node q and p projected on the xy-plane and y is the $high(q)$ value. By definition, in an equidistant grid, all neighbours q of p have the same distance δ to p. That means that they all have x-coordinate δ in the slope diagram.

$U(p)$ is in this case defined by the lowest y-coordinate of all neighbours of p. Since the y-coordinate is given by the $high(q)$ values, we are searching for the neighbour with minimal $high(q)$ value (see Figure 2.6 for an example). We will use this fact and refer to this node(s) as $\mathbf{q_{slopepoint(p)}}$ and we will call the according minimal $high(q)$ value $\mathbf{sdValue(p)}$, which represents the maximal elevation for which a neighbour can win the steepest-neighbour contest. A point p is now always returned by the EXPAND function if the expanded point q' can win the steepest-neighbour contest from $q_{slopepoint(p)}$, and will never be returned if the expanded z'-value of q' is higher than $sdValue(p)$. This property is used in the proof given in Chapter 3. We can write the expand function for an equidistant grid as given below.

Algorithm EXPANDEQ(q, z)
1. $S \leftarrow \varnothing$
2. **for** all neighbours $p \in N(q)$
3. **do if** $high(p) \geq z \wedge sdValue(p) \geq z$
4. **then** $e \leftarrow \max(z, low(p))$
5. $S \leftarrow S \cup (p, e)$
6. **return** S

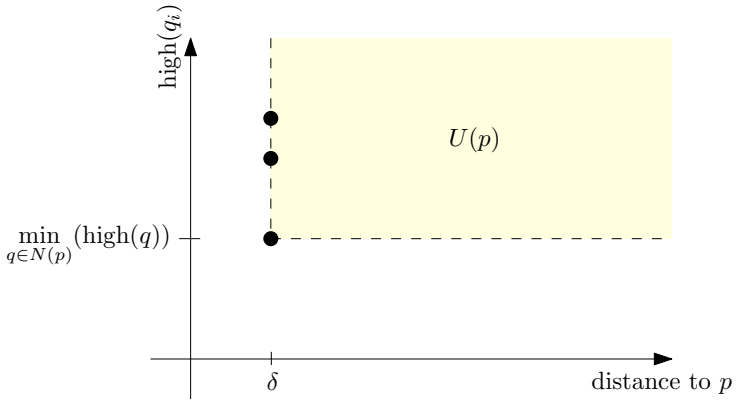


Figure 2.6: Example of a slope diagram in an equidistant grid

CHAPTER 3

Disconnected Persistent Watersheds

As noted in Appendix *B* in Driemel et al. [DHLS11], there exist terrains with disconnected persistent watersheds for certain points in that terrain. In this chapter, we will explain why this example of a disconnected persistent watershed exists and show that we can also find an example for equidistant grids.

We will first explain the cause of the disconnection in the example of Appendix B of [DHLS11]. The example given in the paper can be found in Figure 3.1. In Figure 3.2, the slope diagram of node d is drawn.

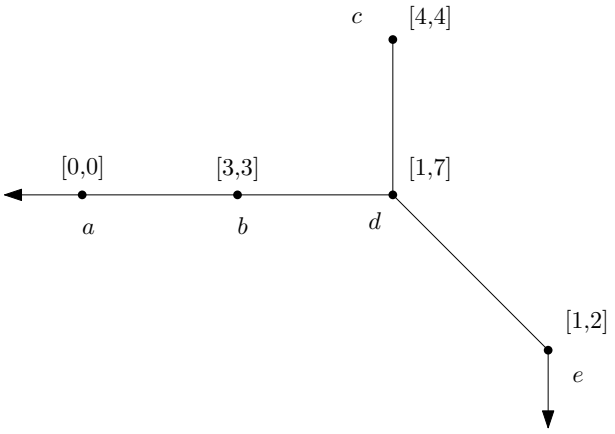


Figure 3.1: Example of a disconnected persistent watershed on a regular terrain

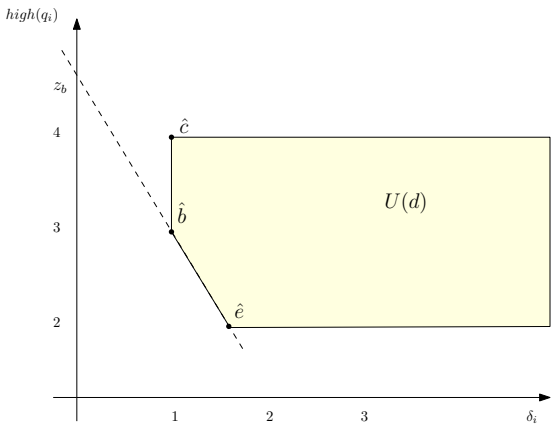


Figure 3.2: Slope diagram of node p of the example in Figure 3.1

In this example, the persistent watershed of node e is the set $\{c, e\}$, two vertices that are disconnected in the given graph. The reason that these two vertices are disconnected can be traced when walking through the steps of the watershed definitions by Driemel et al. [DHLS11]. The persistent watershed of e is given by

$$\mathcal{W}_{\cap}(e) := \left(\mathcal{W}_{\cup}^{\setminus e}((\mathcal{W}_{\cup}(e))^c)\right)^c.$$

That means that the e-avoiding potential watershed of the elements that are not in the potential watershed of e, contains all points except for e and c. To be more precise, this set must contain d and not c and e to disconnect the two nodes in the persistent watershed. The reason why d is in the potential watershed of e, is because there is a setting such that water can flow from d to e (when

$elev_R(d) \geq elev_R(e)$). Also for c there is a possibility to flow to e. However, apparently, there is no setting possible for node c such that water (if leaving c) does not arrive in e, while for node d there must be such a setting possible. The reason that this is the case can be traced back to the shape of the slope diagram of d. Because d has a neighbour that is further away from d than both b and c, the slope diagram gives a minimum elevation of z_b (see Figure 3.2) for which b can be the steepest descent neighbour of d. This elevation however, is higher than the possible elevation of c. Therefore, it is possible for water from d to leave the potential watershed of e, whereas it is not possible for water from c to do so.

Now, we will take a look at the case of an equidistant grid.

Given an equidistant terrain $\mathcal{T}$ and a point q, we want to show that the persistent watershed of q in $\mathcal{T}$ can be disconnected. In that case, we have a persistent watershed of at least two components, S_1 and S_2. We assume w.l.o.g. $q \in S_1$, and we take a point $p \in S_2$ for which there exists at least one realization R_a with a flow path such that water leaves p by flowing to a neighbour $r \in (\mathcal{W}_{\bigcup}(q) \backslash S_2)$. This point r is in the potential watershed but not in the persistent watershed of q and is also part of a flow path in G connecting p and q in at least one other realization $R_b \in \mathcal{R}_{\mathcal{T}}$ (by definition of the potential watershed). It is trivial to see that there is at least one such a pair (p, r).

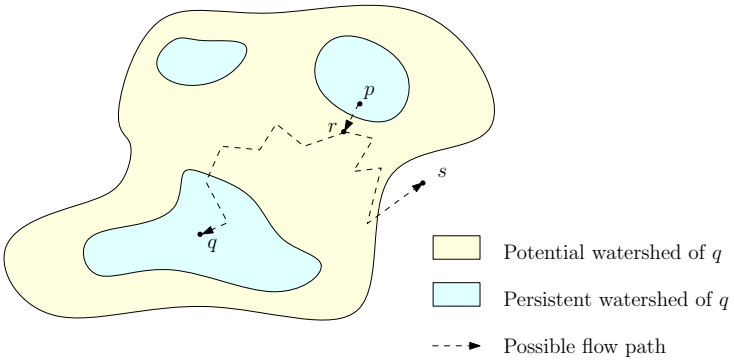


Figure 3.3: Sketch of the situation used to prove that there exist disconnected persistent watersheds in equidistant grids.

By definition of a persistent watershed, $\mathcal{W}_{\bigcap}(q) := \left(\mathcal{W}_{\bigcup}^{\backslash q}((\mathcal{W}_{\bigcup}(q))^c)\right)^c$, we know that r is part of the q-avoiding potential watershed of the complement of the potential watershed of q. Therefore, there must exist a flow path in at least one realization $R_c \in \mathcal{R}_{\mathcal{T}}$ for which water flows from r to a point s in the set of points $(\mathcal{W}_{\bigcup}(q))^c$. Note that $R_a \neq R_c$ and $R_b \neq R_c$, otherwise water could flow from p to r and from r to s, thus creating a flow from p to s; given that p is in the persistent (and not in the q-avoiding potential watershed) watershed of q, there can be no such realization. Note that for each realization for which water flows from r outside the potential watershed, the elevation of r has such a value that r can never be the steepest descent neighbour of p. The situation is sketched in Figure 3.3.

For each point g in the terrain $\mathcal{T}$, we are given a minimum elevation of $low(g)$ and a maximum elevation of $high(g)$. Furthermore, $sdValue(g)$ gives the value of the minimal maximal $high(g')$ values of all $g' \in N(g)$. There are only two possibilities for how r is added to the q-avoiding potential watershed when computing this watershed using the algorithm as described in Driemel et al. [DHLS11]. We denote with $n(r)$ the first neighbour of r that caused r to be enqueued with the lowest enqueued elevation, and we use $elev(n(r))$ to denote this elevation with which the expand step for $n(r)$ was executed.

- A point r is enqueued with elevation greater than $high(p)$ and is therefore added to the potential watershed. If this would be the case, we have $\max(elev(n(r)), low(r)) > high(p)$, as

calculated in line 4 in Algorithm 1. Since there exists a realization (R_a) for which water flows from p to r, we know that $low(r) \leq high(p)$ and thus it must be that $elev(n(r)) > high(p)$. However, we know that $sdValue(r) \leq high(p)$, and thus when $n(r)$ is expanded with elevation greater than $high(p)$, $n(r)$ can never be r's steepest descent neighbour. Therefore, point r can never be enqueued with elevation greater than $high(p)$ in this q-avoiding potential watershed, leading to a contradiction. That means that this point r can not be in the q-avoiding potential watershed and thus this situation does not apply for the point r we are looking for.

- A point r is enqueued with elevation smaller or equal to $high(p)$.
  Given that water can not flow from p to r in the q-avoiding realizations, r must be enqueued with an elevation value, denoted as $elev(r)$, greater than $sdValue(p)$. So we get $sdValue(p) < elev(r) \leq high(p)$. As presented in Figure 3.4, such a setting is indeed possible and therefore, also in equidistant grids the persistent watershed of a point can be disconnected.

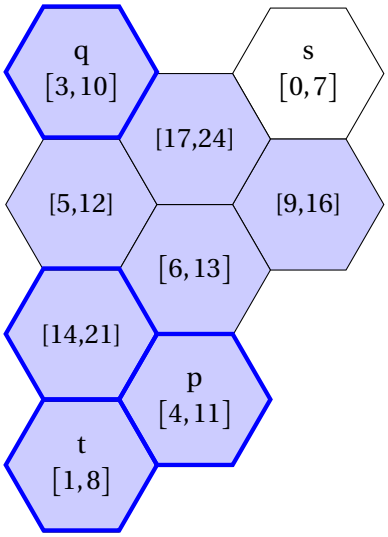


Figure 3.4: Example of a disconnected watershed in an equidistant terrain, with equal imprecise ranges.

In Figure 3.4, node s is 'guarded' by two nodes. Water can only flow to s if the elevation of the nodes connected to these guarding nodes is larger than or equal to the lowest elevation of the guarding nodes. The lowest possible elevation is 9. However, in that case node t on the last row with a range of $[1,8]$ makes sure that water of its two neighbours flows to t itself, since it will be the steepest descent neighbour with a maximum elevation of 8. For the nodes that have no dark blue border, realizations are possible where each node has at least an elevation of 9 and water can flow to s. Therefore, these nodes do not lie in the persistent watershed of q, causing the persistent watershed of q being disconnected. Note that in this example all boundaries of the ranges consist of different values, such that the case of flow to a neighbour with the same elevation does not cause the model to disconnect the persistent watershed.

CHAPTER 4

Watersheds in the MFD Model

The definitions and algorithms for computing watersheds by Driemel et al. [DHLS11] are based on the SFD (Single Flow Direction) approach.This chapter extends the watershed definitions and accompanied algorithms by Driemel et al. to the case of the MFD (Multiple Flow Direction) model. We will show that with some non-trivial adaptations the same algorithms can be used to calculate the defined watersheds for the MFD approach. Note that the in this chapter proposed definitions and algorithms are designed for all network models.

4.1 New watershed definitions

The difference between SFD and MFD is that in the first model, water only flows from a point p to the steepest descent neighbours, whereas in the second model water flows to all neighbours who do not lie higher than p itself. That means we have to adapt the definitions for the watersheds as given in Chapter 2.

For the MFD method the watershed of a point q in a realization R is defined, equal to the SFD method, as the union of nodes that flow to q in R, that is

$$\underset{MFD}{\mathcal{W}_R}(q) := \{p : p \vec{R} q\}.$$

Similarly, we define the discrete watershed of a set of nodes Q in this realization as $\underset{MFD}{\mathcal{W}_R}(Q) := \bigcup_{q \in Q} \underset{MFD}{\mathcal{W}_R}(q)$.

4.1.1 Potential watersheds

The definition of the potential watershed of a set of points Q, is equivalent for both flow methods:

$$\underset{MFD}{\mathcal{W}_\cup}(Q) = \bigcup_{R \in \mathcal{R}_T} \underset{MFD}{\mathcal{W}_R}(Q).$$

This is the set of points for which there exists a flow path to a node in Q. In Section 4.2.1, we see that the algorithm for the potential watershed in the SFD model only needs a minor adaptation to find the potential watershed for the MFD model.

4.1.2 Q-avoiding potential watersheds

The second definition we have to consider is the definition of the Q-avoiding potential watershed of a set S, which needs a little more attention. Since we only use the Q-avoiding potential watershed to find the persistent watershed of Q, we assume, as we did for the SFD method, that also for the MFD method the set S is the complement of the potential watershed of Q. Driemel et al. define the Q-avoiding potential watershed of S as the set of points that can escape from flowing to a point in Q by leaving the potential watershed of Q. In this definition, a point p is able to *escape* from (the potential watershed of) Q when there exists at least one realization where each flow path in which p occurs, flows to a point $s \in S$ without visiting a point $q \in Q$. Local minima, that can not escape the potential watershed of Q, but which do not always flow to a point in Q, are considered not to be part of the Q-avoiding potential watershed. Although this definition seems a little off in some situations, as we saw in the previous chapter with the disconnected persistent watersheds, we will first give a definition that is based on this definition by Driemel et al., followed by a definition of the persistent watershed. In Section 4.2.3, we will give a second definition of the Q-avoiding potential watershed as we would propose it and explain how the algorithms are affected by this new definition.

One important observation for the case of the MFD flow method, is that a point p in a realization R can be contained in multiple flow paths. That means we cannot use the original definition of Q-avoiding potential watersheds by Driemel et al., which considers each flow path separately. This would enable the possibility of including points that have a possible flow to a point in S, while still flowing to a point in Q in another flow path of the same realization. We only want to include those points p that can escape the potential watershed without leaving any water behind in the potential watershed. That means we are looking for points p for which there exists a realization for which all flow paths in which p occurs, will escape the potential watershed by flowing to any of the points $s \in S$, without entering a point $q \in Q$. Flow paths ending up in a local minimum in the potential watershed that do not directly lead to a point in Q are considered as not to be Q-avoiding, according to the definition by Driemel et al.

Let us introduce the notion of $\mathcal{R}_{\mathcal{T}}(\pi)$, which represents the set of realizations for which flow path π exists. We can now define the Q-avoiding potential watershed in the MFD setting:

$$\underset{MFD}{\mathcal{W}_{\bigcup}^{\setminus Q}}(S) = \bigcup_{R \in \mathcal{R}_{\mathcal{T}}} \{p : \forall\, \pi \in \Pi(R),\, \pi \ni p\ \exists\, s \in S\ \forall\, q \in Q : p \xrightarrow{\pi} s \wedge q \notin [p,s]\}.$$

Note that the above definition does not hold when S contains points from inside the potential watershed of Q.

4.1.3 Persistent watersheds

Using the new definition for the Q-avoiding potential watershed, we can reuse the definition of the persistent watershed for the SFD method:

$$\underset{MFD}{\mathcal{W}_{\bigcap}}(Q) := \left(\underset{MFD}{\mathcal{W}_{\bigcup}^{\setminus Q}}\left(\left(\underset{MFD}{\mathcal{W}_{\bigcup}}(Q)\right)^{c}\right)\right)^{c},$$

which is the set of nodes that have at least one realization for which a flow path exists to a point $q \in Q$, and for which in all other realizations the water can not leave the potential watershed of Q .

4.2 Watershed algorithms for the MFD method

Based on these new definitions, we are able to construct algorithms that compute the three types of watersheds for us. The potential watershed computation is done in a similar fashion as described

by Driemel et al., using a different algorithm for the EXPAND function, and is described first. We continue with the algorithm for computing the Q-avoiding potential watershed, where more work was needed to obtain the correct results. The algorithm for computing persistent watersheds is a simple combination of the previous two algorithms and is presented as well. We conclude this section with a short evaluation of the existing definitions.

4.2.1 Potential watersheds

The algorithm that computes the potential watershed of a set Q in the SFD setting, can be adapted to the MFD setting by relaxing the constraints of the steepest-neighbour setting. In the algorithms described by Driemel et al., a slope diagram for point p is used to determine whether a point q' can be the steepest descent neighbour of p. In the case of the MFD model, we need to determine whether there is a realization possible such that q' has a slope ≤ 0 with point p. Therefore, instead of constructing the slope diagram with all $high(q)$ values, we only need to verify whether p can be set to an elevation greater or equal to the elevation z' that q' was enqueued with. If it is possible to set the elevation of a neighbour p to an elevation higher or equal than z', enqueue p with the minimal elevation such that this requirement holds.

The algorithm for the potential watershed of a set Q does not need any other changes, only the expand function is adapted. This results in the algorithms as described by POTENTIALWSMFD and EXPANDMFD.

Algorithm POTENTIALWSMFD(Q)
1. For all $q \in Q$: Enqueue (q,z) with key $z = low(q)$
2. **while** the Queue is not empty
3. **do** (q',z') = DequeueMin()
4. **if** q' is not already in the output set
5. **then** Output q' with elevation z'
6. Enqueue each $(p,z) \in$ EXPANDMFD(q',z')

Algorithm EXPANDMFD(q,z)
1. $S \leftarrow \varnothing$
2. **for** all neighbours $p \in N(q)$
3. **do if** $high(p) \geq z$
4. **then** $e \leftarrow \max(z, low(p))$
5. $S \leftarrow S \cup (p,e)$
6. **return** S

The POTENTIALWSMFD algorithm computes the points and minimal elevation of those points for which a flow path exists to (at least) one of the points in Q. The proof for the correctness of the algorithm for computing the potential watershed of a set Q given by Driemel et al. [DHLS11] still applies, since all points are enqueued with their minimal elevation such that there exists a flow path from p to q in at least one of the possible realizations.

4.2.2 Q-Avoiding potential watersheds

We can calculate the set of points that have a flow path leaving the potential watershed of Q, without visiting any of the vertices of Q on the escape route, by using the algorithm for calculating the Q-avoiding potential watershed for the SFD method, with the EXPANDMFD algorithm. This combination is captured in the Q-AVOIDINGPOTWSMFDA algorithm. This algorithm calculates the points that have a flow path leaving the potential watershed while not visiting a point in Q. In the case of

MFD, however, multiple flow paths can leave a certain point. That means that for another flow path in the same realization, it is still possible that water visits a point in Q or gets stuck in the potential watershed. Therefore, this algorithm does not compute the Q-avoiding potential watershed, but returns a superset of the actual Q-avoiding potential watershed.

Algorithm Q-AVOIDINGPOTWSMFDA(S, Q)
1. For all $s \in S$: Enqueue (s, z) with key $z = low(q)$
2. **while** the Queue is not empty
3. **do** (s', z') = DequeueMin()
4. **if** s' is not already in the output set $\wedge s' \notin Q$
5. **then** Output s' with elevation z'
6. Enqueue each $(p, z) \in$ EXPANDMFD(s', z')

One would think that points that are output with a lower elevation compared to the potential watershed output belong to the Q-avoiding potential watershed; If they can leave the potential watershed by a flow path with a lower height than the minimal flow height to a point in Q in the potential watershed, we know that there exists a realization for which water from this node will never end up in Q – trivially, since the points in the potential watershed had minimal elevation for which a flow path existed to a point in Q. However, it is possible that these flow paths divide into multiple flow paths and one of these flow paths ends up in a local minimum within the potential watershed. By definition, the points for which this holds do not belong to the Q-avoiding potential watershed. Let us denote the set of points that have no flow path $\pi \in \Pi(\mathcal{R}_\mathcal{T})$ leaving the potential watershed by Q'. These are the points in the potential watershed minus the points calculated by the Q-AVOIDINGPOTWSMFDA algorithm. For our final Q-avoiding potential watershed, we do not only need to look at flow paths ending up in Q, but to flow paths ending up in Q'.

Another observation in the case of MFD is that enqueueing a point p at its highest elevation minimizes the number of possible flow paths towards p (flowing to or visiting p on its path); one could see this as a blocking mechanism to prevent flow paths to pass this point. This observation is the basis of our actual computation. We use this to calculate the so-called *peak avoid watershed* of a set Q' and a set Q_A, where $e_p^{Q_A}$ represents the enqueued elevation for which p was added to Q_A and Q' is defined as $\mathcal{W}_\cup(Q)\backslash Q_A$:

$$P_A(Q_A) = \{q' : q' \in Q'\} \cup \left\{p : p \in Q_A \wedge \exists_{\pi^+ \in \Pi(R^+_\mathcal{T}) \wedge q' \in Q' \wedge p' \in N(p)} : (p \underset{\pi^+}{\rightarrow} p') \wedge (p' \underset{\pi^+}{\rightarrow} q') \wedge (e_p^{Q_A}(p) \geq high(p'))\right\}.$$

This set contains all points that cannot escape the potential watershed based on the enqueued elevation with which they were added to Q_A, a set containing potential Q-avoiding candidates.

The pseudo-code is given by the PEAKAVOIDMFDWS algorithm. The set Q_A represents the set of points with a flow path to outside the potential watershed. The set Q' is defined as $\mathcal{W}_\cup(Q)\backslash Q_A$, and contains only (maybe not all) points that can not leave the potential watershed without leaving water behind, since the points in $\mathcal{W}_\cup(Q)\backslash Q_A$ cannot leave $\mathcal{W}_\cup(Q)$at all, except in a realization in which also a flow path to a point $q \in Q$ exists — by definition of Q_A. The algorithm enqueues all points in Q' at their highest elevation and adds these points to the peak persistent watershed, after which it expands the flow search. We know, since Q_A is an upper bound on the points that can leave the potential watershed, that for a point in Q', water will always stay in the potential watershed or flow via a point in Q before leaving the potential watershed. Therefore, we can enqueue these nodes at highest elevation, without influencing the flow from Q' to outside the watershed, while blocking as much flow to Q' as possible — one can not enqueue any point of Q' at a higher elevation. We have a set of enqueued points p with enqueued elevation e_p, forming the peak persistent watershed so far, and we expand the search for a neighbour $n(p) \in Q_A$ based on the following condition:

- If $n(p) \in Q_A \wedge e^{Q_A}_{n(p)} \geq e_p$, water from $n(p)$ is never able to escape from Q' when it wants to leave the potential watershed, since p (and all other points on the flow path to a point $q \in Q'$) is enqueued at its highest elevation and $n(p)$ has to be set at a higher elevation to be able to leave the potential watershed, which means that water will flow to Q' after all. Therefore, we can enqueue $n(p)$ at the highest elevation to block as much flow to Q' as possible.

Note that a point $n(p) \in Q_A$ with $e^{Q_A}_{n(p)} < e_p$ is not added to the peak avoid watershed, since it still has an opportunity to leave the potential watershed without leaving water behind. We do not have to check explicitly whether we add points from outside the potential watershed to the peak avoid watershed; if there would be a flow possible to a point in Q, these points would belong the the potential watershed as well, and since all points in $Q' \backslash Q$ are enqueued at their highest elevation, we also know that there will be no flow to one of these points as well – they were enqueued with a height smaller equal to their maximal height and did not enable flow from points outside the potential watershed then, therefore not enabling this flow now either.

Algorithm PEAKAVOIDMFDWS(Q', Q_A)
1. For all $q \in Q'$: Enqueue (q, z) with key $z = high(q)$
2. **while** the Queue is not empty
3. **do** (q', z') = DequeueMin()
4. **if** q' is not already in the output set
5. **then** Output q' with elevation z'
6. Enqueue each $(p, z) \in$ EXPANDHIGHAVOIDMFD(q', z', Q_A)

Algorithm EXPANDHIGHAVOIDMFD(q, z, Q_A)
1. $S \leftarrow \varnothing$
2. **for** all neighbours $p \in N(q)$
3. **do if** $\neg((p, e) \in Q_A \wedge e < z)$
4. **then** $S \leftarrow S \cup (p, high(p))$
5. **return** S

The algorithm for calculating the Q-avoiding potential watershed of a set of points S is given by the Q-AVOIDINGPOTWSMFDB algorithm. We start with the set Q_{A0}, the potential Q-avoiding potential watershed as calculated by the Q-AVOIDINGPOTWSMFDA algorithm – the superset of Q-avoiding potential watershed candidates, and Q', as defined above. We iteratively strip the set Q_{A0} by removing nodes that will end up in Q' after all. We start by calculating the peak avoid watershed of Q and Q_{A0}. We now have a set of points P_{A0} for which water will flow to Q or partly ends up in the potential watershed when trying to leave the potential watershed. We fix the elevations of these points at their highest possible elevation and recalculate the set of potential Q-avoiding candidates $Q_{A(i+1)}$. With this new set $Q_{A(i+1)}$ we again calculate the peak avoid watershed, but now for $\mathcal{P} \backslash Q_{A(i+1)}$. The above process (enqueue at highest elevation, check whether still escape possible) repeats itself until $Q_{Ax} = Q_{A(x+1)}$. Using this repetition, we filter all flow paths that will lead to a local minimum within the potential watershed or flow paths that will visit a point in Q. The points that eventually end up in Q_{Ax} represents the set of points of the final Q-avoiding potential watershed. These points are able to leave the potential watershed without leaving water behind in the potential watershed.

In set notation, the algorithm computes the following sets. We start with the set containing all points with a possible escape to outside the potential watershed:

$$Q_{A0}(S, Q) = \left\{ p : \exists_{\pi \in \Pi(\mathcal{R}_\mathcal{T}) \wedge s \in S} \forall_{q \in Q} : \neg(p \xrightarrow[\pi]{} q) \wedge (p \xrightarrow[\pi]{} s) \right\}.$$

Note that this definition assumes that S does not contain any node from the potential watershed of Q, which is the case in the computations for which we use this definition.

Algorithm Q-AVOIDINGPOTWSMFDB$(S, Q, \mathcal{P})$
1. $i \leftarrow 0$;
2. $Q_{Ai} \leftarrow$ Q-AVOIDINGPOTWSMFDA(S, Q)
3. $P_{Ai} \leftarrow$ PEAKAVOIDMFDWS$(\mathcal{P} \backslash Q_{Ai}, Q_{Ai})$
4. **repeat** For all $(p, e) \in P_{Ai}$: Update (temporarily) $low(p) \leftarrow e$
5. $\quad i \leftarrow i + 1$
6. $\quad Q_{Ai} \leftarrow$ Q-AVOIDINGPOTWSMFDA$(S, P_{A(i-1)})$
7. $\quad P_{Ai} \leftarrow$ PEAKAVOIDMFDWS$(\mathcal{P} \backslash Q_{Ai}, Q_{Ai})$
8. **until** $Q_{Ai} == Q_{A(i-1)}$
9. For all $(p, e) \in Q_{Ai}$: Output p with elevation e

Next, we calculate each iteration i the peak avoid watershed regarding the set Q_{Ai}, starting with Q_{A0} which is defined. Let us denote the set $\mathcal{W}_{\cup}(Q) \backslash Q_{Ai}$ by Q'_i. Note, that by definition $p \underset{\pi}{\rightarrow} p$. The peak avoid watershed of an iteration i can now be rewritten to:

$$P_{Ai}(Q_{Ai}) = \{q' : q' \in Q'_i\} \cup \left\{p : p \in Q_{Ai} \wedge \exists_{\pi^+ \in \Pi(R^+_{\mathcal{T}}) \wedge q' \in Q'_i \wedge p' \in N(p)} : (p \underset{\pi^+}{\rightarrow} p') \wedge (p' \underset{\pi^+}{\rightarrow} q') \wedge (e_p^{Q_{Ai}} > high(p'))\right\}.$$

This gives us the points p that can not escape from flowing to a point in Q'_i with their potential watershed escape elevation $e_{qi}(p)$ when all points on the flow path are set to their maximum elevation. The definition of the sets Q_{Ai}, $i > 0$, can now be given by:

$$Q_{Ai}(S, P_{A(i-1)}) = \left\{p : \exists_{\pi \in \Pi(\mathcal{R}_T) \wedge s \in S} \forall_{q \in P_{A(i-1)}} : \neg(p \underset{\pi}{\rightarrow} q) \wedge (p \underset{\pi}{\rightarrow} s)\right\}.$$

When $Q_{Ax} = Q_{A(x+1)}$, the points for which a realization exists in which they can escape from the potential watershed, without occuring in any flow path that ends in the potential watershed or visits a point in Q, the Q-avoiding potential watershed, can be described by the set Q_{Ax}.

There are three ingredients of the Q-avoiding potential watershed algorithm that we need to prove correctness of:

1. The Q_{Ai} sets are actual supersets of the Q-avoiding potential watershed.
2. The points in the peak avoid watershed can be set to their highest elevation for new calculations, while still computing the correct Q-avoiding potential watershed.
3. At the moment that $Q_{Ax} = Q_{A(x+1)}$, we have found our Q-avoiding potential watershed, Q_{Ax}.

Let us start with the second ingredient, since we use this fact when calculating new Q_{Ai}.

Lemma 4.2.1 *Points for which in all realizations at least one of the flow paths ends up in Q' can be set to their highest elevation for new calculations, while still computing the correct Q-avoiding potential watershed.*

Proof We are given that the points computed by the peak avoid watershed algorithm are the points for which in any realization water will end up in a point in Q'. It is safe to set the *low* values for these points p to their *high* values, for new Q-avoiding calculations, since we know that water will end up in Q' for these nodes after all and the highest possible elevation for these points blocks as many flow paths flowing to or visiting p, as possible. Points in the final Q-avoiding potential watershed should have a realization for which the flow paths to outside the potential watershed do not cross any of the nodes in this peak avoid watershed, because if they do, water will end up in Q or in a local minimum and the point should not be in the Q-avoiding potential watershed. Therefore, we can safely update the *low* values to their *high* values.

Given that we can enqueue the points in the peak avoid watershed at their highest elevation for new calculations, we continue with the first ingredient.

Lemma 4.2.2 *The Q_{Ai} sets are actual supersets of the Q-avoiding potential watershed.*

Proof We will prove this lemma by induction. We know, by definition, that our initial set Q_A is a superset of the actual Q-avoiding potential watershed; this set contains all points that have at least one flow path leaving the potential watershed. Now, let us assume that the set Q_{Ai} is a superset of the Q-avoiding potential watershed. Now, we calculate the peak avoid watershed, which returns the set P_{Ai} of points that will have a flow path to a point in $Q' = \mathcal{W}_{\cup}(Q) \backslash Q_{Ai}$ for their enqueued elevation in Q_{Ai} when all points in Q' are set to their highest elevation. By definition, these are the points from which some flow will always remain in the potential watershed. We enqueue these points at their highest elevation, before calculating the new set $Q_{A(i+1)}$. The points in Q_{Ai} that are no longer possible to escape had a flow path via a point that was added to the peak avoid watershed in the last iteration and is now set to its highest elevation, since otherwise the flow path would still be possible. By definition, this point would visit a point that does not belong to the Q-avoiding potential watershed, and therefore this point also does not belong to the Q-avoiding potential watershed. For the points that are still in $Q_{A(i+1)}$ we know that there still exists a flow path to outside the potential watershed. That means that the sets Q_{Ai} indeed are supersets of the Q-avoiding potential watershed.

Given the fact that the Q_{Ai} sets are supersets, we now need to prove that we indeed find the correct Q-avoiding potential watershed.

Lemma 4.2.3 *After the last iteration of the algorithm, at the moment that $Q_{Ax} = Q_{A(x+1)}$, we have found our Q-avoiding potential watershed, Q_{Ax}.*

Proof At the moment the algorithm does not add any new point to the peak avoid watershed, we have found our Q-avoiding potential watershed. The peak avoid watershed checks for the potential Q-avoiding candidates whether there still exists a flow path to any point in Q', with all $q' \in Q'$ set to their highest elevation. If such a flow path still exists, the point for which this is the case is added to the peak avoid watershed and the algorithm needs another iteration. If there does not exist any such a flow path, we know that for each of the points $p \in Q_{Ax}$ there exists a realization for which water can escape the potential watershed without leaving any water behind, namely the realization in which all points in Q' are set to their highest elevation and all point q' to their enqueued elevation as enqueued in Q_{Ax}. Exactly for these enqueued elevations the peak avoid watershed is determined and no flow paths to a point in Q' was found. That means that we now have found the set of points that can escape the potential watershed without leaving any water behind: Q_{Ax}, the Q-avoiding potential watershed.

All together, this gives us that the above algorithm indeed calculates the Q-avoiding potential watershed as defined.

4.2.3 Persistent watersheds

The complement of the Q-avoiding potential watershed then yields the persistent watershed. The complete algorithm for calculating the persistent watershed in the MFD setting can then be described by the algorithm PERSISTENTWSMFD.

4.2.4 Running times

The running times of the POTENTIALWSMFD algorithm and the Q-AVOIDINGPOTWSMFDA algorithm are the same as for the SFD method, $O(n \log n)$. The proof is similar to the proof by Driemel et al.

Algorithm PERSISTENTWSMFD(Q)
1. $\mathcal{P} \leftarrow$ POTENTIALWSMFD(Q)
2. $\mathcal{A} \leftarrow$ Q-AVOIDINGPOTWSMFDB($\mathcal{P}^c, Q, \mathcal{P}$)
3. Output $\mathcal{A}^c$

[DHLS11]. The same proof can also be applied to the PEAKAVOIDMFDWS algorithm, since the only difference is that nodes are enqueued at their highest elevation, still leaving the three types of nodes during the graph search. The bottleneck in the computations is the Q-AVOIDINGPOTWSMFDB algorithm, caused by the repeat-until statement. We know that this loop can only be executed a number of times of the size of the potential watershed of Q – each iteration at least one point must be added to the peak avoid watershed, otherwise the Q_A set remains the same. That means that the algorithm has a worst case running time of $O(size(\mathcal{W}_{\bigcup_{MFD}}(Q)) \cdot n \log n) = O(n^2 \log n)$. For the persistent watershed, we need to calculate both the potential and the Q-avoiding potential watersheds to find the set of points of the persistent watershed, by taking the complement of the second set. In total, calculating the persistent watershed with algorithm PERSISTENTWSMFD costs $O((n \log n) + (n^2 \log n) + (n \log n)) = O(n^2 \log n)$ time.

4.2.5 Evaluation

The above definitions and arguments are based on the definition as proposed by Driemel et al., which state that points that can not leave the potential watershed without flowing through a point in Q always belong to the persistent watershed. This definition however, may be a little off in some cases, as mentioned before. For a local minimum for which all neighbours can leave the potential watershed without flowing to a point in Q, it is not always as intuitive that this local minimum belongs to the persistent watershed of Q — as with the disconnected persistent watersheds in the previous chapter for example. Solely looking at the possible elevation values and realizations, one would indeed be able to conclude that, if water leaves this local minimum, it would flow to a point in Q. However, in the realization where all neighbours can escape the potential watershed, it is more likely that water collects in this local minimum and when fully filled, it is also possible to leave the potential watershed by flowing to one of the neighbours that was already able to leave the potential watershed. To take this situation into account in the case of the SFD or the MFD method, the definition of the Q-avoiding potential watershed could be adapted to a set of points that can escape the potential watershed without flowing to Q, together with points that can indirectly escape the potential watershed without flowing to Q, for example the points for which the neighbour with the minimum elevation can escape from the potential watershed. One of the adaptations for the Q-avoiding potential watershed algorithm then involves changing the set Q' for which the PEAKAVOIDMFDWS algorithm is invoked, in such a way, that it should not contain every point that has no possible flow path outside the potential watershed of Q. Further research should be done to find out whether this definition and a suitable algorithm realizing this definition gives the desired results.

All computations in the rest of this report are based on the first definition and the algorithms as described above.

CHAPTER 5

NP-hardness in the Slope-Bounded Model

In this chapter, we will show that it is NP-hard to decide for the slope-bounded model whether there exists a setting such that water could flow from a node s to a node t. We prove NP-hardness for the slope-bounded model together with the flow model where water will flow to the steepest descent neighbour of a node, the SFD method. The reduction is from 3-SAT; with a 3-CNF formula as input, with n variables and m clauses. Based on the formula, we construct a slope-bounded model. We will first explain the construction method that is used for the proof, then we give the details of the construction and finally we will prove the correctness. At last, we show that this problem is still NP-hard in the special model instance of an equidistant grid.

5.1 General idea of the construction

Before we give the proof, let us formulate the decision problem and give a brief description of the 3-SAT method we use to prove the NP-hardness.

> **Problem:** Given a slope-bounded model as defined in Section 2.2.2, does there exist a realization of this model such that in this realization, water will flow from a given node s to a given node t?

To prove that this problem is NP-hard, we find a polynomial-time reduction from 3-SAT to the above problem. If we can find the answer to the above problem in polynomial time, then we can also find the answer to all 3-SAT problems in polynomial time, which is generally believed to be impossible. So by giving a valid reduction, we can show that the above problem is at least as hard as the 3-SAT problem. Given that 3-SAT is NP-complete, showing the reduction gives us that our decision problem above is NP-hard.

3-SAT is a special instance of the satisfiability problem. In the case of 3-SAT, we are given a formula in conjunctive normal form, where in each of the m clauses three of the n variables appear (for instance $((x_1 \vee \neg x_2 \vee \neg x_3) \wedge (x_1 \vee x_2 \vee x_4) \wedge \dots)$). The general idea of the NP-hardness construction is to use the variables and clauses from the 3-CNF formula to create a slope-bounded model, such that a truth assignment to the variables corresponds to a realization in the slope-bounded model. We want to prove that *if and only if* in such a realization water flows from a certain starting vertex s to a certain target vertex t, all clauses in the 3-CNF formula are satisfied. For the realizations that do not correspond to a truth assignment of the 3-CNF formula, we want that water will never flow from s to t.

The construction of the slope-bounded model consists of two basic elements: switch and connector gadgets. Since water will only flow to the steepest descent neighbour, we can create local minima in the model – where water gets stuck – by making sure that all neighbours have a positive slope with a certain node, and thus a higher elevation than that node.

Waterfall stairs The final construction of the slope model is based on a number of stairs connected to each other. Each stair represents one of the m clauses of the formula and each stile one of the n variables. The stairs and stiles are connected such that the flow of water is influenced based on the choice of the slopes in the model, which are dependent on the truth value of the variable representing the stile (see Figure 5.1). The vertices in this Figure are examples of distribution and collection nodes as explained in the next paragraphs. Each stile has a number of distribution nodes and a number of collection nodes, to influence the water flow based on the truth assignment of the variable that is represented by that stile. The edges are the connections between these nodes, which is also explained in more detail below, and allow water to flow to the following node for certain realizations of the model. If and only if water can flow down via all m stairs, the 3-CNF formula can be satisfied. The exact setting and connections are explained in detail in the next subsection. We will first explain the basic elements of the construction. Since each stile represents the combination of a variable and a clause, we use the subscripts x_{ijk} to denote the k'th number of the variable of type x belonging to the stile that represents the combination of the i'th variable and the j'th clause.

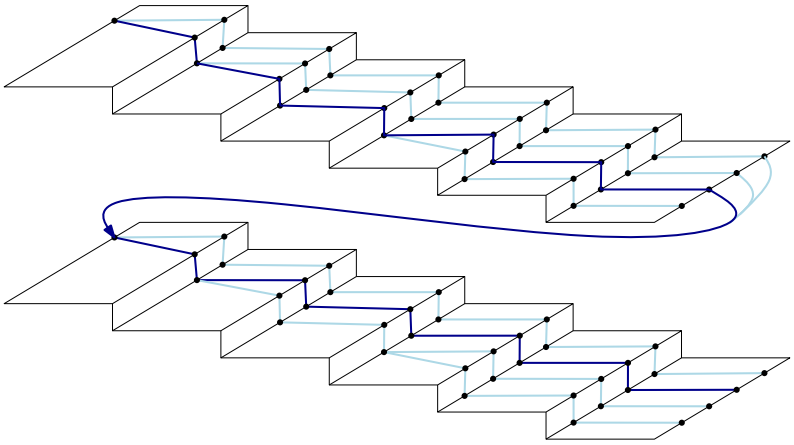


Figure 5.1: The water flow is influenced based on the slope choices in the model. Sketch of two stairs and the possible routes of the water flow.

Distribution nodes Since we want to translate an assignment of the 3-CNF formula to a realization in our slope model, we have to keep track of the result of a clause in one way. To accomplish this, we use different distribution nodes along the way of a stair case. Each stile has four different distribution nodes, located on the left of the stile and all on the same height but on a certain distance from each other. The exact details are discussed in the part where we discuss all the details of the construction. The four distribution nodes each represent a truth assignment to the variables of the clause we visited so far. If we have not crossed a variable of the clause yet, water flows through the first distribution node. After the first variable of the clause has been traversed, water flows either through the first or the second distribution node, or it stays behind in a local minimum, based on the truth value of this first variable. After the second variable, water flows through one of the four distribution nodes. One of the four then represents the truth setting where both values were false. By the information the distribution nodes give us – which truth assignment it represents – we can influence the flow path of the s-t flow: when we come across a variable of the clause, the slope value representing the truth

value of that variable determines to which collection node – see the next paragraph – the water is diverted.

Collection nodes To differentiate the realizations that represent truth assignments from the other realizations, we introduce collection nodes. A distribution node is connected to three collection nodes, located on the right of the stile on a lower elevation as the distribution node and all three with a different height compared to each other and on a certain distance from each other. These collection nodes represent the true, false or confused state of the variable of the current stile in the realization corresponding to the evaluation of the variable in that clause – the truth assignment, the false assignment and the *confused state*, when it is not clear which truth value is represented by the realization. Based on the elevation of the distribution node, water flows to one of the collection nodes. Based on the collection node where water has arrived, we can decide to which of the four distribution nodes of the next variable water should be traversed. In this way, we can decide at the stile of the third variable of a clause whether the assignment of the variables would yield true or not. If not, the flow of water is halted by creating a local minimum. Otherwise, the slopes (and thus assignment of the variables) give a valid truth assignment to that clause and water can flow to the next stairs (see Figure 5.2 for a top view sketch). In this Figure, the stair case is unfolded and the stiles can be described as the larger rectangles of the two types of rectangles. As mentioned, the distribution nodes are on the left of this rectangle and the collection nodes are on the right. Water flows via an edge (in the direction of the arrows) from a distribution node to one of the collection nodes. The way the water flow is influenced is determined by the connector gadget, as explained below.

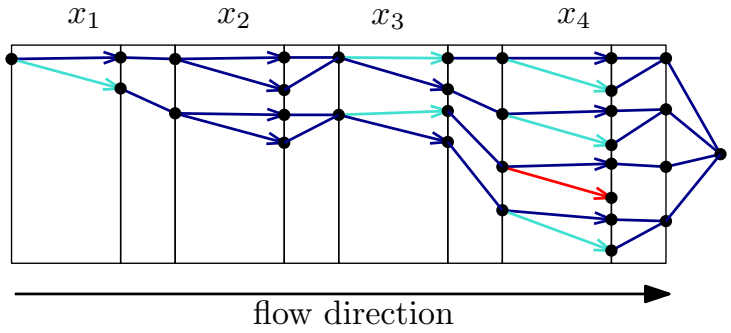


Figure 5.2: The water flow depends on the truth values of the variables (the slopes in the model). Top view showing the flow paths for the clause $(x_1 \vee \neg x_3 \vee x_4)$ on the stair case with four variables. Light blue shows when water flows to the false collection nodes, the red edge means that after this the clause yields false, and thus a local minimum is created. For simplicity, the confused states are left out.

Switch gadgets A switch gadgets enables a water flow to switch between the connected collection nodes. A basic switch gadget consists of one incoming node – the distribution node s, three collection nodes (t, f and c), and at least one helper node h that is connected to a *control edge* e. The setting is sketched in Figure 5.3. When the helper node is set to its highest possible elevation, the water will flow to the f node. Is it set to its lowest possible elevation, water will flow to the t node. For the settings in between, water will flow to the c node.

The reason that this is possible is the fact that the three nodes can be placed at fixed distances with respect to the s node and fixed elevations with respect to the complete model. Furthermore, the helper node defines the elevation of the s node. Choosing the right values for the distances of the edges and the fixed elevation of the f,c and t nodes, enables the switch gadget to divert the flow to the collection node that is the steepest neighbour depending on the elevation of the s node. The

switch gadget used in our construction is a slightly adapted one; with four distribution nodes that are connected to two helper nodes, h_{ij1} and h_{ij2}, with slope 0, making sure they all have the same elevation in every realization (see Figure 5.4 for a top view). When the variable in the clause is negated, we switch positions of the f and t collection nodes, such that the flow is diverted based on the correct truth assignment.

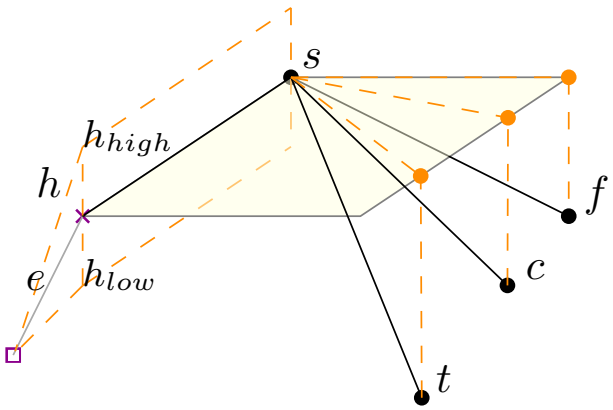


Figure 5.3: Basic switch setting

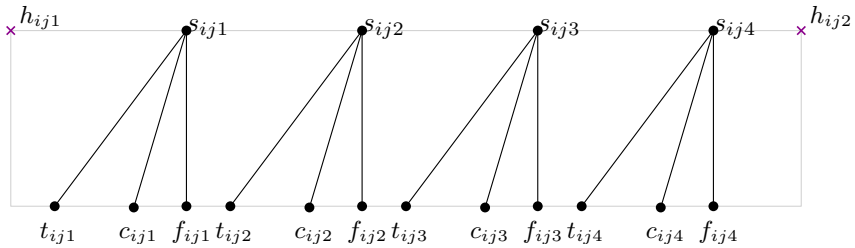
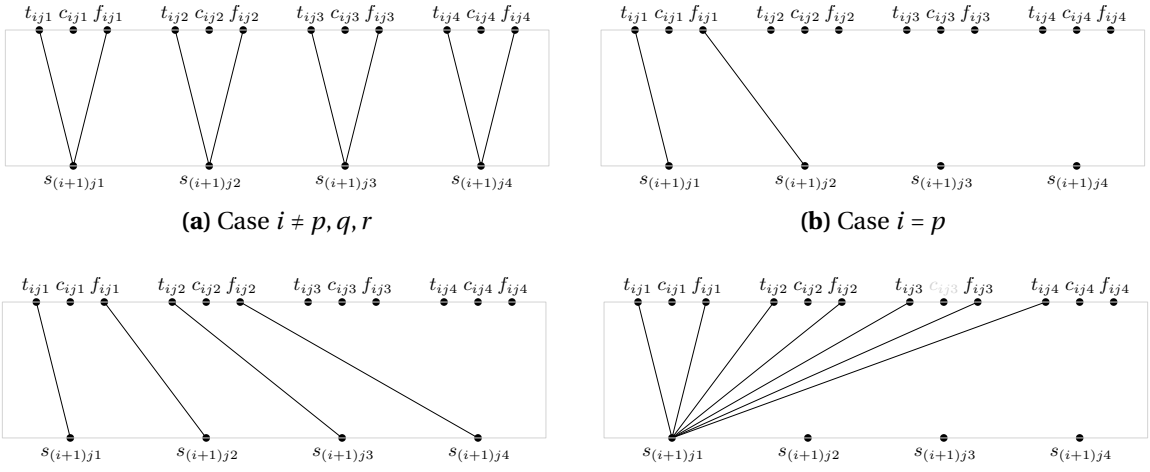


Figure 5.4: Sketch of a switch gadget used to control the flow based on the height of the distribution nodes.

Connector gadgets To connect two switch gadgets, we use a connector gadget. These are the smaller rectangles of the two types of rectangles in Figure 5.2. A connector gadget connects the collection nodes with the distribution nodes of the next switch gadget, based on certain criteria. Since we have to do with a 3-CNF formula with the three variables v_p, v_q and v_r $(p > q > r)$, in our construction the connection is based on the variable for which the switch gadget operates. Since a clause consists of three variables, we need four different connector gadgets, as presented in Figure 5.5. Between two switch gadgets of the same clause, the connector gadget decides which edges are introduced. These gadgets make sure that the water flow is adapted, based on the variable of the current stile. A connector gadget has four different distribution nodes $s_{ij1}, s_{ij2}, s_{ij3}$ and s_{ij4}. At the beginning of a stair case, the water flows in the first distribution node s_{ij1} of the first stile of the stair case. For each stile of a variable that is not part of the clause of the current stair case, the connector gadget of Figure 5.5a connects the switch gadget of the current stile to the switch gadget of the next stile. This connector gadget diverts the water to the distribution node with the same number as it arrived on the previous stile, since the variable has no influence on the result of the clause. That means that water always flows via the first distribution node s_{ij1} when entering the switch gadget of the first variable v_p of the clause. After the switch gadget of v_p, the connector gadget shown in Figure 5.5b is connected. After the flow is switched, water can only be in one of three collection nodes x_{ij1} $(x \in \{t, f, c\})$ connected to the distribution node s_{pj1}. Based on the collection node the water arrives at, the water is

diverted to either of the two distribution nodes $s_{(p+1)j1}$ or $s_{(p+1)j2}$ of the next stile. Here, the first distribution node $s_{(p+1)j1}$ represents the state for which the variables until now all are assigned true, and the second distribution node $s_{(p+1)j2}$ represents the state that of all variables traversed so far, exactly one variable of the clause was set to false. So, after the first variable is traversed, it is possible that water flows via the first or the second distribution node. Using the connector gadgets of case (a) for the variables that are not part of the clause, water arrives in either the first or second distribution node at variable v_q. Therefore, case (c), the connector gadget for the second variable, deals with the collection nodes connected to distribution nodes one and two. Again, the diversion is based on the collection node water flowed to. Now, for the next stiles on the stair case, the first distribution node s_{ij1} ($i > q$) represents the state for which the variables until now all are assigned true, and the second distribution node s_{ij2} ($i > q$) represents the state that of all variables traversed so far, the first variable of the clause was set to false. The third distribution node s_{ij3} ($i > q$) represents the state that of all variables traversed so far, the second variable of the clause was set to false. The last and thus the fourth distribution node s_{ij4} ($i > q$) represents the state for which the variables until now all are assigned false. Again, using the connector gadgets of case (a) for the variables that are not part of the clause, water arrives in one of the four distribution nodes at variable v_r. Now, the connector gadget as shown in Figure 5.5d is connected. For the last variable v_r, each of the seven assignments making the clause evaluate to true is diverted to the first distribution node $s_{(r+1)j1}$ of the next stile. The collection node that represents the situation where all variables evaluate to false, is changed to a local minimum, by not connecting it to any distribution node of the next stile.

In this way, we can guide the water down the stairs based on the slope values of the nodes the water flows through. The slope values, which are directly related to the slope of the control edge, representing the truth setting of a variable, decide in what collection node the water arrives. The combination of the collection node water arrived at, whether or not the variable occurs in the clause of the current stair case and whether or not the variable is negated, decides to which distribution node of the next stile water is sent. By connecting each collection node to at most one distribution node, and by making sure the distribution nodes of a next stile have a lower elevation than the collection nodes of the current stile, water is always sent to the connected distribution node. This distribution node is the only lower situated neighbour that is connected to the collection node and therefore will always win the steepest neighbour contest. In this way, a slope bounded model can represent a 3-CNF formula.



(a) Case $i \neq p, q, r$ **(b)** Case $i = p$

(c) Case $i = q$ **(d)** Case $i = r$

Figure 5.5: The four different settings of the connector gadget for a variable v_i (1 <= i <= n) for a 3-CNF formula with the clause variables v_p, v_q and v_r. Note that the settings to connect a switch gadget to the sink of a column are not shown here.

5.2 Details of the construction

With the global idea of the construction and the basic elements in mind, we will now give the exact details of the construction. Note that we can enforce a certain elevation onto some of the points in the construction by creating a sort of framework with selected slope values. We will use this to connect the stiles of one variable to each other and to construct the stairs. As already explained, the construction consists of m stairs with each n stiles.

We want to enforce that the first stile of the $(j+1)$'th clause has a lower elevation than the last stile of the j'th clause (as sketched in Figure 5.6). In that case, the water flow is always able to proceed to the next stairs. Furthermore, we want to place the collection nodes of a distribution node on a fixed elevation relative to the lowest elevation of the distribution node, such that the slope chosen for the variable – which on his turn influences the elevation of the distribution node – indeed causes the right switching behavior as explained above. We use helper nodes to create the bounding box to enforce these conditions.

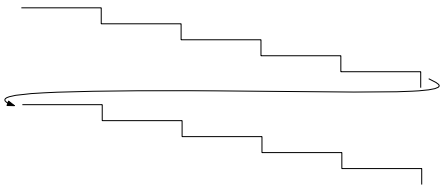


Figure 5.6: Water can always flow to the next clause.

The construction is based on a set of nodes with coordinates, a set of edges between these nodes and a slope range defined for each of these edges. We will give the relative coordinates of the nodes within the gadgets, the coordinates of the helper nodes and the settings of the control edges. Then, we will explain how the gadgets are connected and how the exact coordinates can be obtained, based on a gadget's row and column.

We will give the sizes and distances in the xy-plane, and for the edges introduced we will also define the corresponding slope range. If the slope of an edge only consists of one possible value, we also call it the slope value of that edge. The top view of the construction is sketched in Figure 5.7. The S boxes represent the switch gadgets, the C boxes the connector gadgets and the F boxes are connector gadgets that are adapted to divert the flow to the next stairs if necessary. The F box contains the connected gadget that would be selected as described above, with the adaption that all edges that would connect a collection node to a distribution node $s_{(i+1)jk}$ ($k \in \{1,2,3,4\}$) of the next gadget are now replaced by edges connecting to one 'fake' distribution node d_j that is connected to the first distribution node of the next stairs $s_{1(j+1)1}$. For the last stile of the last stairs, the 'fake' distribution node is t.

Every row of the construction represents a clause of the 3-CNF formula, every column a variable of the alphabet used by the formula. A row consists of n switch gadgets. If a variable is not contained in the clause represented by the row, or if the variable occurs non-negated in the clause, the switch gadget as shown in Figure 5.8 is placed on the stile. Otherwise, the t_{ijk} and f_{ijk} collection nodes are switched. The switch gadgets have size five by one and are placed with distance one between them to create the stairs. The connector gadgets only add edges to connect the switch gadgets, so they introduce no extra nodes.

In Figure 5.9, the edge lengths in the xy-plane between the distribution node and the connected collection nodes are given, together with the distances in the xy-plane between the collection nodes. In Figure 5.10, the slope diagram of one distribution node of a switch gadget is given. Because of symmetry, we can use the same lengths and distances for each of the four distribution nodes. Based on the distances as given in Figure 5.9, we can find values for the elevations of the nodes in the slope diagram such that the setting indeed diverts the water to the right collection node in the right setting.

Using these values, we can calculate the elevations and slopes for the nodes within a gadget. Assume node s_{ijk} has coordinates $(x_s, y_s, [5,6])$, where $[5,6]$ means that the node can have an elevation between 5 and 6. Now we can define the coordinates of the connected collection nodes;

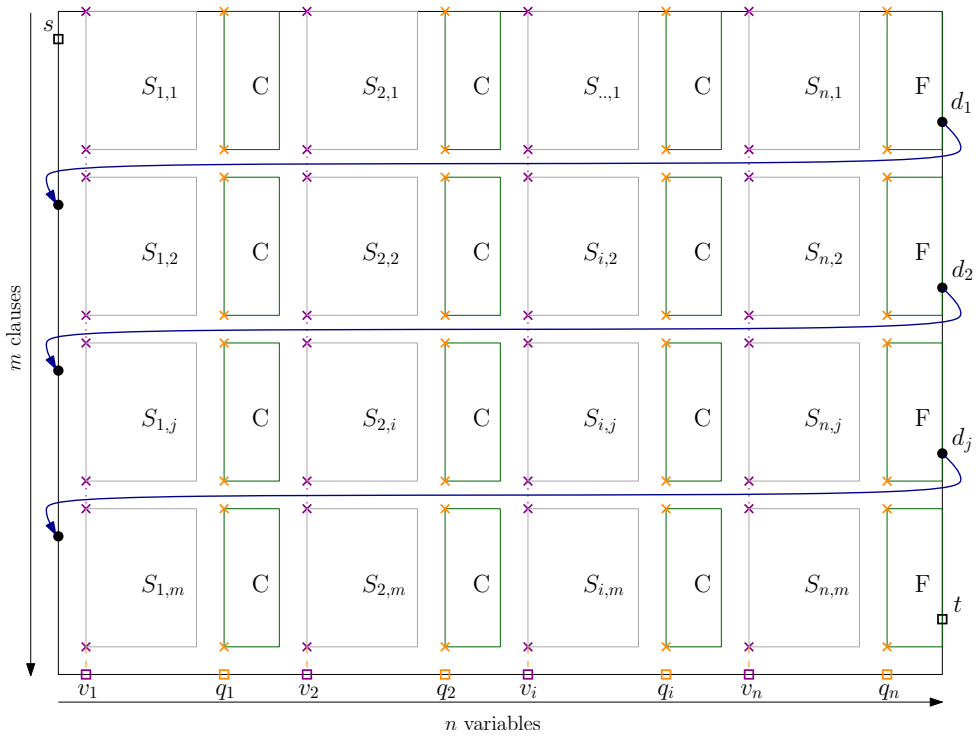


Figure 5.7: Sketch of the total construction, viewed from top.

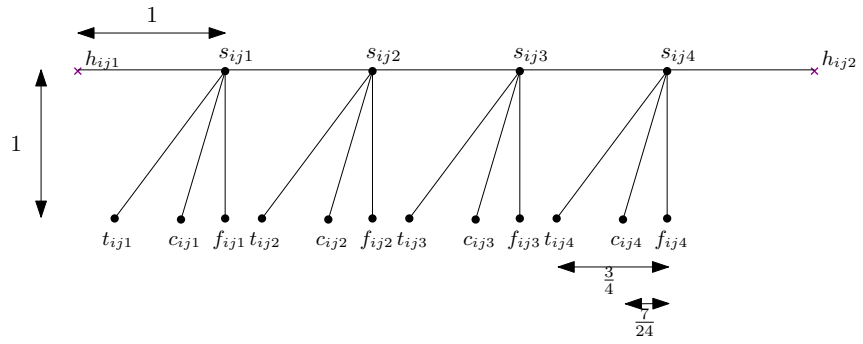


Figure 5.8: Distances in the switch gadget (top view).

$t_{ijk} = (x_s + 1, y_s - \frac{3}{4}, 0.1312)$ and $c_{ijk} = (x_s + 1, y_s - \frac{7}{24}, 0.8)$ and $f_{ijk} = (x_s + 1, y_s, 1)$. When s_{ijk} has a height below $5\frac{1}{5}$ the flow is directed to flow gate t_{ijk}. When the height is in range $[5\frac{1}{5}, 5\frac{4}{5})$, the flow is directed to the confused flow gate, c_{ijk}. And when the height is higher or equal to $5\frac{4}{5}$, the flow is directed to flow gate f_{ijk}.
The slope ranges between the distribution nodes and collection nodes are:

- For the edge $(s_{ijk}, t_{ijk}))$, the elevation difference is within $[-5.8688, -4.8688]$. Divided by the distance of the edge in the xy-plane, $\frac{6}{5}$, this gives us a slope range $slopeRange((s_{ijk}, t_{ijk})) = [-4.8907, -4.0573]$.
- For the edge $(s_{ijk}, c_{ijk}))$, the elevation difference is within $[-5.2, -4.2]$. Divided by the distance, $\frac{25}{24}$, this yields $slopeRange((s_{ijk}, c_{ijk})) = [-4.992, -4.032]$.
- Since the distance for the edge (s_{ijk}, f_{ijk}) is one, the slope range is equal to the elevation difference: $slopeRange((s_{ijk}, f_{ijk})) = [-5, -4]$.

The elevation differences between the collection nodes and the distribution nodes of the next stile are for edge $(t_{ijk}, s_{(i+1)j1})$ a difference between $[-1.1312, -0.1312]$. For edge $(c_{ijk}, s_{(i+1)j1})$ in the

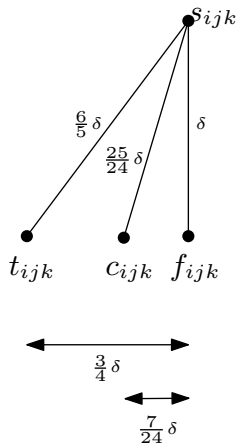


Figure 5.9: Distances in the switch gadget for one distribution node.

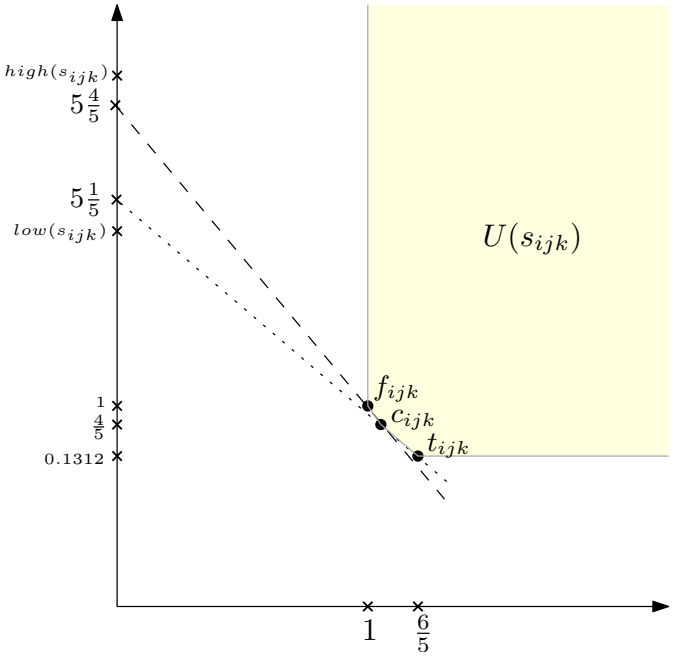


Figure 5.10: Slope diagram of the distribution nodes s_{ijk}.

range of $[-1.8,-0.8]$, and for edge $(f_{ijk}, s_{(i+1)j1})$ within $[-2,-1]$. The slope ranges can be calculated by dividing the elevation difference range by the distance from the collection node to the connected distribution node in the xy-plane.

We will make sure (using the helper nodes) that each stile has a height of 6 and the top of the next stile starts at the bottom of the current stile, making sure that these slopes can be obtained. The slope of the edges (h_{ij1}, s_{ij1}), (s_{ij1}, s_{ij2}), (s_{ij2}, s_{ij3}), (s_{ij3}, s_{ij4}) and (s_{ij4}, h_{ij2}) are set to slope value 0, making sure they will be at the same elevation. We will also connect the collection nodes of the switch gadget in such a way, that the elevations are exactly as explained.

We can now connect multiple switch gadgets to construct the stairs for a clause. Now, we want to connect the stairs of each clause such that the setting indeed realizes the slope model to prove the NP-hardness. The stairs of each clause is placed within distance one from the stairs of the next clause and for each pair helper nodes $(h_{ij1}, h_{i(j+1)2})$ an edge is added with slope value $s = (-6n-1)$ such that the stile of the next stairs is placed n stile-heights plus one lower. The situation is sketched in Figure 5.11. The helper node of the last clause h_{im1} is connected to the variable node v_i, the purple box as sketched in Figure 5.7, with a control edge as explained earlier. The variable node is placed distance 1 from the helper node and the control edge (v_i, h_{im1}) has slope range $[0,1]$. In this way, all stiles of one variable are raised when this control edge is set to a higher slope. The slope of the edge connecting the fake distribution node d_j to the first distribution node of the next stairs $s_{1(j+1)1}$ is set to $(-1/(2n))$, since a stile is size 1 and between two stiles also 1 space is added.

The front side view how the variable nodes and collection helper nodes q_i are connected in the total framework is depicted in Figure 5.12. The numbers on the edges are the slope values for each edge. The collection helper nodes q_i, depicted by the orange boxes, connect to the corresponding collection nodes of the i'th variable with fixed slopes only. That means that the collection nodes will have a fixed height compared to the height of the framework. The slopes between all the distribution nodes of on variable are also fixed, except for the control edge connected to the variable node v_i, which can increase the height of the variables' distribution nodes all by a height within 0 and 1.

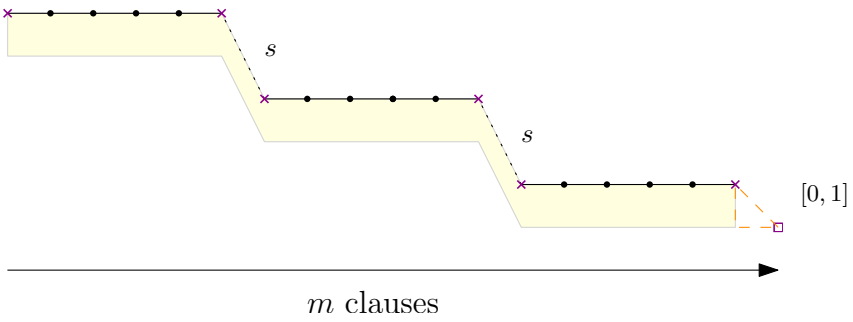


Figure 5.11: Side view connecting the switch gadgets for variable j.

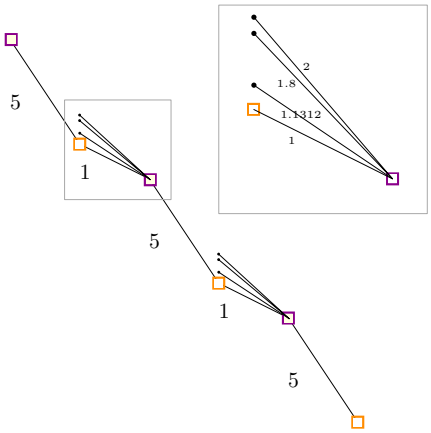


Figure 5.12: Slopes in front view.

5.3 Correctness of the NP-hard reduction

Lemma 5.3.1 *If water flows from s to t in some realization, then there is a truth assignment of the variables of the 3-CNF formula that satisfies the formula.*

Proof Water starts flowing from s, which is in our construction the first distribution node of the first clause, $s_{1,1,1}$. As calculated above, water entering a switch gadget at one of the distribution nodes will either end up in a local minimum or it will enter the next switch gadget in one of the distribution nodes. That means that water from s can only reach t after flowing through *all* switch gadgets. Since all c_{ijk} channels lead to local minima, we know that if there is a flow path from s to t, no switch gadget can be in a confused state. That means that all control edges must have a slope either in $[0, 0.2)$ or in $(0.8, 1]$. If the variable occurs non-negated in the clause, the gadget in the column of that variable leads in the first case to a true state of that gadget (otherwise to a false state). In the second case, the water leaves the gadget as it was in a false state (or a true state when negated). We can now construct a truth assignment to the variables in which each variable is true if the control edge is in the lower range as described above, and false otherwise. Since water flows through each stairs and reaches t, this setting satisfies each clause, and thus the complete 3-CNF formula.

Lemma 5.3.2 *If there is a truth assignment to the variables that satisfies the given 3-CNF formula, then there is a realization of the slope model in which water flows from s to t.*

Proof We set the control edge for variable v_j to 0 if the value was assigned true in the assignment and to 1 if it was assigned false. By construction, in each clause stairs, water from the first distribution node will reach the fake distribution node at the end, and thus, water from s reaches t.

Thus, 3-SAT can be reduced, in polynomial time, to deciding whether there is a realization of the slope model such that water can flow from s to t. Therefore, deciding whether there exists a realization in this slope model such that water flows from s to t is NP-hard.

5.4 NP-hardness in equidistant slope model

The setting described above is constructed based on the influence of the slope diagram of the distribution nodes, where the different distances between the t_{ijk}, c_{ijk} and f_{ijk} nodes decide where the water flows to in the different settings of the distribution node. One may wonder whether it is still NP-hard when all distances between neighbours are the same. We call this setting the equidistant slope model; a slope model where all edges have the same length. Unfortunately, also for this model, deciding whether water flows from a point s to t is still NP-hard. To prove this, we use the same construction as above, replacing the switch gadgets and connector gadgets with gadgets where all edges have the same length, as described below.

Equidistant switch gadgets The switch gadget for the equidistant slope model is sketched in Figure 5.13. To make sure that the setting also works in an equidistant grid where all nodes are connected, the nodes of the gadgets are placed on specific places in the hexagonal grid, such that the blank neighbours can be added with a positive slope such that water can never flow to these nodes. However, since filling in these details for these irrelevant nodes makes the gadget unreadable, we did not include these edges in the Figure. It is easy to verify that it is indeed possible to add edges with positive slopes such that water can never flow to the currently 'unused' neighbours.

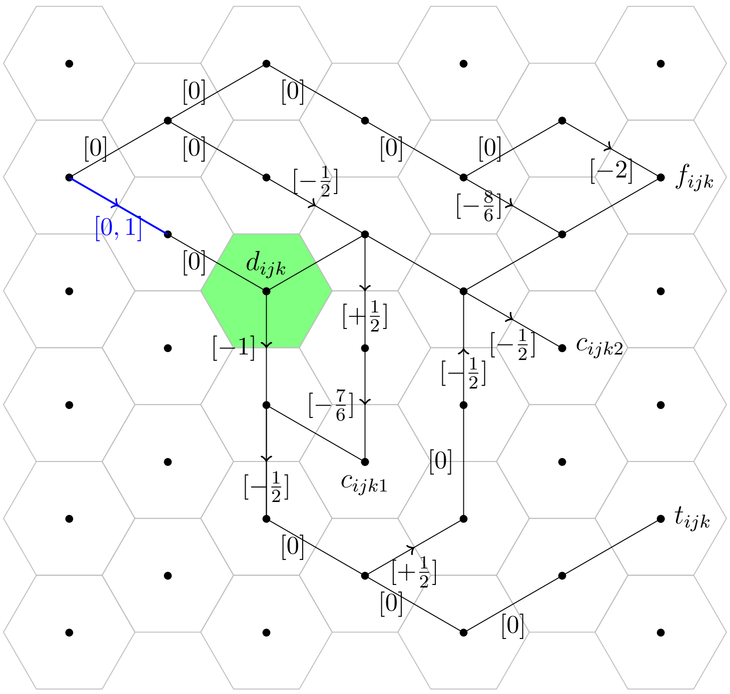


Figure 5.13: General switch gadget for the equidistant slope model

The idea behind this setting is that we create a dependency between the neighbours that is based on the height of the green node – the distribution node. As depicted in Figure 5.13, the elevation of the distribution node depends on the selected slope of the blue edge, the 'control edge'. As already mentioned above, we can set elevations relative to the zero setting of the node, by connecting a node to the ground node (before the control edge influenced the height of the distribution node). Using

this method, we can set several elevations to a fixed value. Other elevations can be set relative to the elevation of the distribution node by setting a fixed slope. In the slope diagram of the distribution node, we will have a neighbour with a fixed elevation, and one with a fixed slope. Depending on the elevation of the distribution node, one of the two nodes is the steepest descent neighbour and water will flow to that neighbour. In this way, we can create a setting where water can flow to different nodes in different settings. We sketched the situation in Figure 5.14 with the slope diagram of the distribution node d_{ijk} and of nodes a and b in Figure 5.15. Note that we only draw the slope diagram for elevations of a node for which water actually flows to that node.

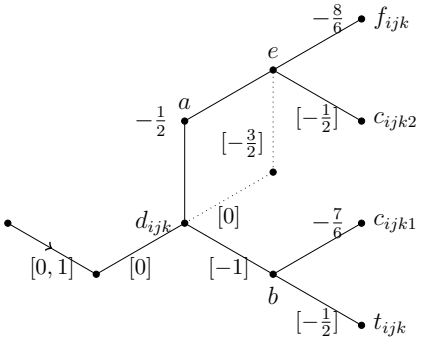
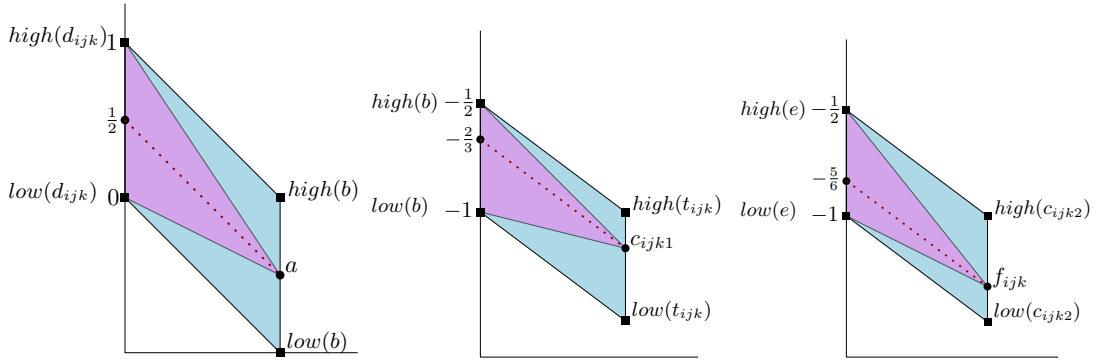


Figure 5.14: Elevations in the equidistant switch gadget

The distribution node d_{ijk} has an elevation between zero and one, based on the slope of the control edge. The only neighbours that have a (possible) lower elevation than d_{ijk} are nodes a and b. As can be seen in Figure 5.15a, a is the steepest descent neighbour for $d_{ijk} \geq \frac{1}{2}$, and b is the steepest descent neighbour for $d_{ijk} \leq \frac{1}{2}$. In the case that $d_{ijk} = \frac{1}{2}$ both nodes get an amount of water. Therefore, we can not distinguish the true and false state of this gadget based on this difference in elevation, and we have to introduce a confused state again. However, since we have two different flows of water after which the confused state should be reached, it is easier to create a setting with two confused nodes. When water arrives in one of these two nodes, the gadget is in a confused state. In the setting we describe here, we have a true-state when the distribution node d_{ijk} has an elevation in the range of $[0, \frac{1}{3})$, a true-ish state when the elevation is $\frac{1}{3}$, a confused state in the range of $(\frac{1}{3}, \frac{2}{3})$, a false-ish state when the elevation is $\frac{2}{3}$ and the gadget will be in the false state when the distribution node has an elevation in the range of $(\frac{2}{3}, 1]$. In the true-ish and false-ish states, water does not only flow to a confused node, but also to either the t_{ijk} or f_{ijk} node. Furthermore, to halt the flow when d_{ijk} has an elevation of $\frac{1}{2}$, causing water flowing to both neighbours, we ensure by this setting that both flows end up in a confused node.

First, we will look at the water flow when $d_{ijk} \in [0, \frac{1}{2}]$. In this case, water flows to node b. As can be seen in Figure 5.15b, node c_{ijk1} will be the steepest descent neighbour of b when b has an elevation between $-\frac{2}{3}$ and $-\frac{1}{2}$. This corresponds to an elevation of node d_{ijk} between $\frac{1}{3}$ and $\frac{1}{2}$. For a lower elevation of node d_{ijk}, water flows via b to node t_{ijk}. Now, let us look at the case when $d_{ijk} \in [\frac{1}{2}, 1]$. In this case, water flows to node c. Since e's elevation is $\frac{3}{2}$ lower than d_{ijk}'s elevation, for $d_{ijk} \in [\frac{1}{2}, 1]$ the elevation of e lies within the range of $[-1, -\frac{1}{2}]$. And with an elevation of $-\frac{1}{2}$ for c, water from node c will always flow to e. Looking at the slope diagram of node e as depicted in Figure 5.15c, we can see that water flows to the f_{ijk} node when $d_{ijk} \geq \frac{2}{3}$ and for $d_{ijk} \in [\frac{1}{2}, \frac{2}{3}]$, water flows to the confused node c_{ijk2}. That means that this setting fulfills the requirements as described and we can distinguish between the true-, false- and confused states of the gadget. When the variable occurs negated in the clause, the labels of the t_{ijk} and f_{ijk} are switched.



(a) Node d_{ijk} **(b)** Node b **(c)** Node e

Figure 5.15: Slope diagrams of the equidistant switch gadget

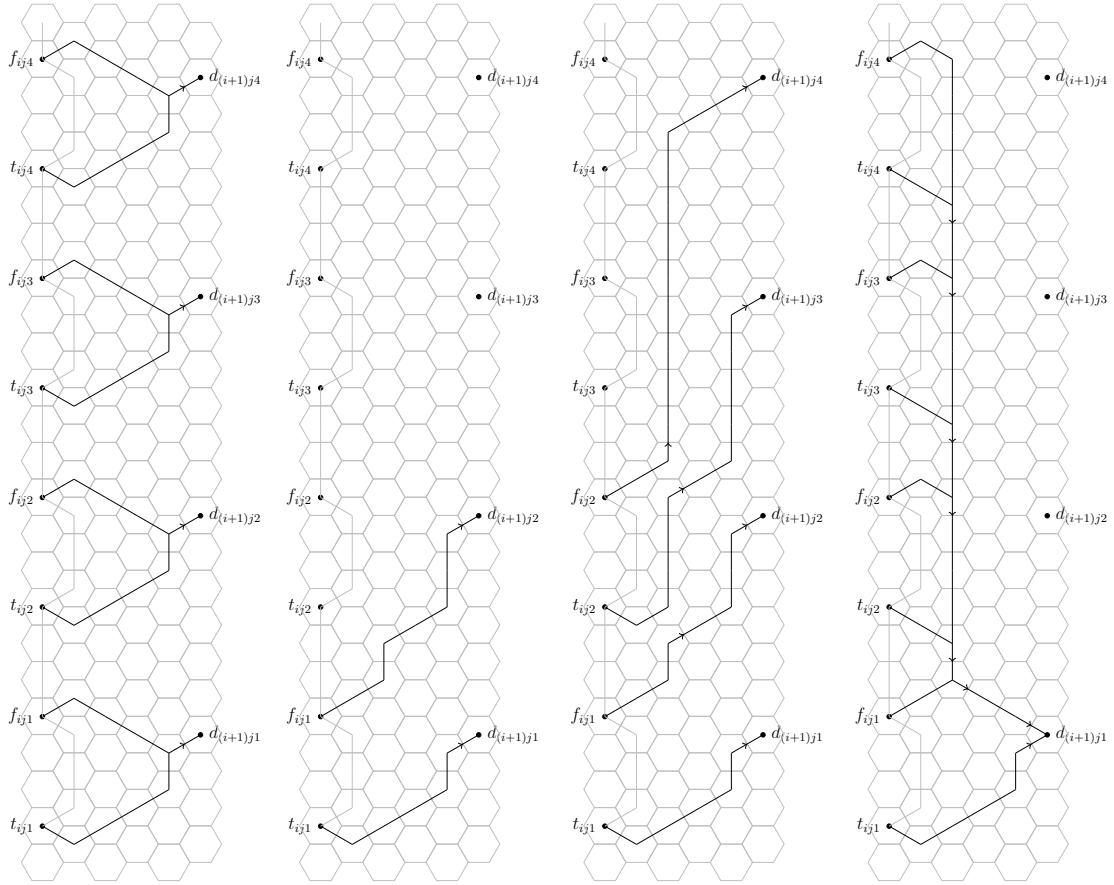
Equidistant connector gadgets The connector gadgets for the equidistant slope model connect the t_{ijk} and f_{ijk} nodes in the same way as the connector gadgets described before, however, now the edges are spread over multiple cells to ensure flow to the right distribution node. The gadgets are sketched in Figure 5.16. Note that we left out the edges between neighbouring nodes, since that only distracts from the actual flow path that is managed by these connector gadgets. The edges with an arrow make sure that water flows to the right neighbour and flows paths can not cross. The actual slopes only have to differ an amount of $\epsilon > 0$, such that the neighbour on the flow path will be the steepest descent neighbour. Overall, the connector gadget will bridge an elevation of 1 before connecting to the distribution nodes. Where the edges are inserted to obtain this total difference in elevation in the gadget, is up to the constructor, as long as the requirements for the flow paths are met.

Total construction For the construction of the stair cases, we connect the switch and connector gadgets as in the general slope model construction, with the edges and slopes as described in this setting. The connection between each stair case is adapted, to make sure that the setting also works in an equidistant grid. We change the setting such that the stairs are connected by a flow path in a planar graph structure, such that edges can not cross. The situation is sketched in Figure 5.17. The blue edges on the bottom of the construction are the control edges for each variable. The orange edges connect the switch gadgets such that the slope of the control edge is propagated to all stiles of the variable. The dark red edges connect the connector gadgets (in particular the t_{ijk} and f_{ijk} nodes) to make sure that we can use the 'zero' height as sketched in the switch gadget above. The black edge at the bottom of the figure regulates the base difference in stiles. The slope differences between the nodes are sketched in Figure 5.18. This determines the edges and slopes to construct the staircase for each clause. By only allowing edges with zero or negative slope, we can make sure that the flow path connecting two stair cases is downwards and that the second stair case is in total below the first one. Furthermore, we can calculate the total elevation difference of the stair case, since this is defined by the construction that connects the control edges at the bottom of the total construction. The stiles of a variable are connected with an edge defining this slope plus one (of the stair case connecting flow path), such that the slope of the control edge is indeed propagated.

5.4.1 Correctness of the NP-hard reduction

Lemma 5.4.1 *If water flows from s to t in some realization of the equidistant slope model construction, then there is a truth assignment of the variables of the 3-CNF formula that satisfies the formula.*

Proof Water starts flowing from s, which is in our construction the first distribution node of the first clause, $d_{1,1,1}$. As calculated above, water entering a switch gadget at one of the distribution nodes



(a) Case $i \neq p, q, r$ **(b)** Case $i = p$ **(c)** Case $i = q$ **(d)** Case $i = r$

Figure 5.16: The four different settings of the connector gadget in the equidistant slope model for a 3-CNF formula with the variables p, q and r. Note that the settings to connect a switch gadget to the sink of a column are not shown here.

will either end up in a local minimum or it will enter the next switch gadget in one of the distribution nodes. That means that water from s can only reach t after flowing through *all* switch gadgets. Since all c_{ijk} channels lead to local minima, we know that if there is a flow path from s to t, no switch gadget can be in a confused state. That means that all control edges must have a slope either in $[0, \frac{1}{3}]$ or in $[\frac{2}{3}, 1]$. If the variable occurs non-negated in the clause, the gadget in the column of that variable leads in the first case to a true state of that gadget (otherwise to a false state). In the second case, the water leaves the gadget as it was in a false state (or a true state when negated). We can now construct a truth assignment to the variables in which each variable is true if the control edge is in the lower range as described above, and false otherwise. Since water flows through each stairs and reaches t, this setting satisfies each clause, and thus the complete 3-CNF formula.

Lemma 5.4.2 *If there is a truth assignment to the variables that satisfies the given 3-CNF formula, then there is a realization of the equidistant slope model in which water flows from s to t.*

Proof We use the total construction as described above in the equidistant slope model setting and we set the control edge for variable v_j to 0 if the value was assigned true in the assignment and to 1 if it was assigned false. By construction, in each clause stairs, water from the first distribution node will reach the fake distribution node at the end, and thus, water from s reaches t.

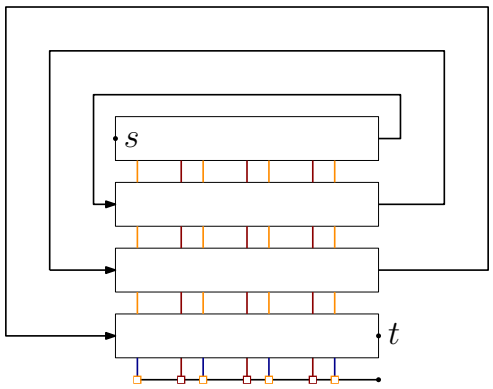


Figure 5.17: Connecting the stair cases in the equidistant slope model

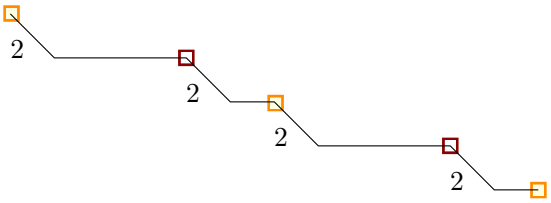


Figure 5.18: Slopes in front view

Thus, 3-SAT can be reduced, in polynomial time, to deciding whether there is a realization of the slope model such that water can from from s to t. Deciding whether there exists a realization in this equidistant slope model such that water flows from s to t is NP-hard.

CHAPTER 6

Implementation

In this chapter we discuss the implementation details of the watershed algorithms as used to obtain the results in the next chapters. Note that all results are based on calculations on hexagonal grid models. We can divide the implementation into two parts: creating the uncertain elevation models, including actions such as deciding the elevation ranges and the coordinates of the data points, and the second part, calculating properties of these uncertain elevation models. Since elevation models are usually very large (several GBs), we need to take memory management into account. The first part of the implementation is described in Section 6.1; this part covers interpolating the elevation values and constructing the models, which can be done by a sequential read of the data and therefore that is all the memory management that is necessary. The code for this first part is written in Python 2.7 using the statsmodels package (http://statsmodels.sourceforge.net). The second part covers the actual calculations on the models and is described in Section 6.2; this part is written in C++ and compiled using the gcc 4.5.1 compiler. All tests are run on a computer with a Linux 2.6.35.14 Kernel, four Intel Core2 Quad Processors each 2.66GHz, a memory of 8GB and a second hard disk of 500GB.

6.1 Creating the uncertainty models

Since most GIS applications use normal square grid models as input models and we want to calculate with hexagonal grid models, we need to translate the existing square grids to hexagonal grid models. Since the input files are very large, we can not simply store all information in memory and then continue with calculating the hexagonal grid information. Therefore, we divide the input file in smaller blocks and examine the results of these blocks one by one. The easiest way to implement this is by reading the minimum number of rows of the original input file into memory for calculating one row of the hexagonal grid. After calculating this hexagonal grid row, we can remove the rows of the rectangular input model that have no use any more and read the lines we need for calculating the next hexagonal grid row. We can continue this until we reach the end of the document. There are several ways to place a hexagonal grid on top of a rectangular grid. In our implementation, the elevation of a cell (a,b) in the hexagonal grid (row a, column b), is determined by the formulae given below, where $rect(x,y)$ refers to the elevation of the cell on the x'th row and in the y'th column in the rectangular grid and ℓ represents the smallest row index in the rectangular grid for which the data is available in memory. By substracting this ℓ we refer to the local rectangular grid, that exists only of the lines read to memory that were necessary for this calculation. We need to distinguish the situations that we are calculating the cells of an even or an odd column in the hexagonal grid, because of the shifted cells.

In Figure 6.1a is shown how we can place a hexagonal grid onto a square grid. Based on this overlay method, we can find the elevation values for the centres of the hexagonal grid cells as follows:

b odd:

$$r_1 = rect(\lfloor (a+1)\cdot\sqrt{3}\rfloor - \ell, \frac{1+3b}{2}) \qquad r_3 = rect(\lceil (a+1)\cdot\sqrt{3}\rceil - \ell, \frac{1+3b}{2})$$

$$r_2 = rect(\lfloor (a+1)\cdot\sqrt{3}\rfloor - \ell, \frac{3+3b}{2}) \qquad r_4 = rect(\lceil (a+1)\cdot\sqrt{3}\rceil - \ell, \frac{3+3b}{2})$$

$$x_1 = r_1 + (\frac{1}{2}(r_2 - r_1))$$

$$x_2 = r_3 + (\frac{1}{2}(r_4 - r_3))$$

First, we interpolate the upper and lower coordinates of the square to get values x_1 and x_2 as depicted in Figure 6.1b. Since the centre of the hexagonal cell lies exactly on the centre of these edges, this is a simple calculation step. Next, we interpolate these x-values. Here, we need to take into account where the centre of the hexagonal grid cell is placed on the line segment (x_1,x_2):

$$hex(a,b) = x_1 + ((\sqrt{3}(a+1) - \lfloor\sqrt{3}(a+1)\rfloor)\cdot(x_2 - x_1)).$$

b even :

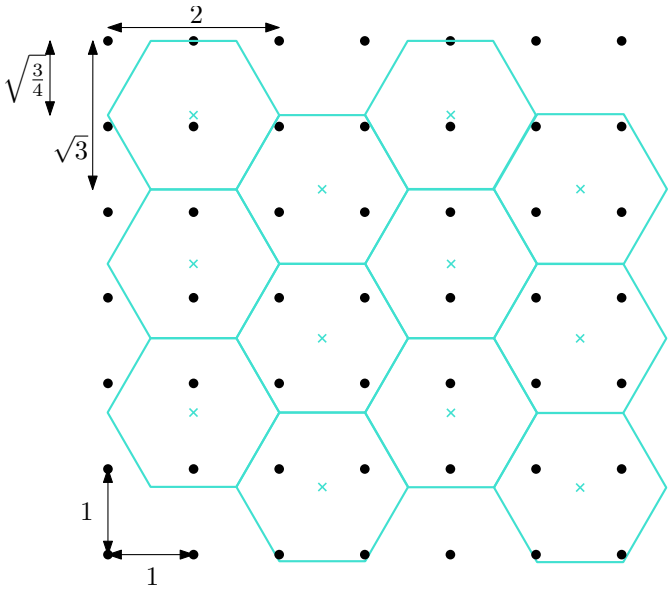
$$r_1 = rect(\lfloor (a+\frac{1}{2})\cdot\sqrt{3}\rfloor - \ell, \frac{2+3b}{2})$$

$$r_2 = rect(\lceil (a+\frac{1}{2})\cdot\sqrt{3}\rceil - \ell, \frac{2+3b}{2})$$

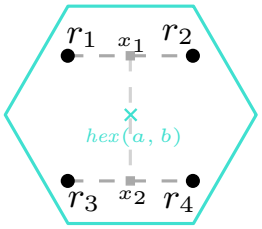
Now, the elevation of the cell in the hexagonal grid model can be calculated with:

$$hex(a,b) = r_1 + ((\sqrt{3}(a+\frac{1}{2}) - \lfloor\sqrt{3}(a+\frac{1}{2})\rfloor)\cdot(r_2 - r_1)).$$

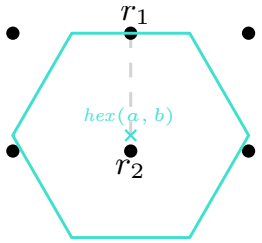
This overlay method will create a $\lfloor\frac{n-1}{\sqrt{3}} - 1\rfloor \times \lfloor\frac{2m}{3} - 1\rfloor$ hexagonal grid from a $n \times m$ rectangular grid.



(a) Overlay of a hexagonal grid onto a square grid



(b) b odd



(c) b even

Figure 6.1: Interpolating a square grid to a hexagonal grid for the hexagonal grid column number b.

In our implementation, the input (a square grid file) is an ASCII file with on the first four lines the north, south, east and west coordinates of the terrain, on the fifth line the number of rows n and on the sixth line the number of columns m of the input data, followed by n lines containing the elevation values in row major order from left to right, as space-delimited floating point numbers. The output (a hexagonal grid file) is a text file with on the first line "HEX", to denote that this file contains information on a hexagonal grid, followed by the number of rows, columns and then the elevation values as described above. When a data point does not exist in the original input, the output file has '−1' on that place in the file, the no-data value.

Below, the pseudo-code for this interpolation method can be found. The algorithm reads as many lines as needed to compute the elevations of a new hexagonal grid row. The function *isOdd*(a) returns whether value a is odd. The function *split*(i) splits a line from the input data file into an array of the values that are separated by spaces. The function array.*append*(x) appends the input x to the array, array.*pop*(i) removes the i'th index from the array.

Algorithm INTERPOLATESQTOHEX($fileIn, fileOut$)
Input: File *fileIn* with a rectangular grid input
Output: File *fileOut* containing the interpolated values of the hexagonal grid
1. Read file *fileIn* until line 4
2. Write line HEX to *fileOut*
3. $a, hexRows, hexCols \leftarrow 0,0,0$
4. $rectGrid \leftarrow [\,]$
($*$ $rectGrid[a,b]$ contains the elevation value for row a, column b $*$)
5. $smallestL, largestL \leftarrow 0,-1$
6. $smallestLforA, largestLforA \leftarrow \lfloor \sqrt{3} \cdot a \rfloor, \lceil \sqrt{3}(a+\frac{3}{2}) \rceil$
7. $nrRows \leftarrow$ number on fourth line
8. $nrCols \leftarrow$ number on fifth line
9. Write line $nrRows$ to *fileOut*
10. Write line $nrCols$ to *fileOut*
11. **for** line $i \leftarrow 6$ **to** $nrRows+5$
12. **do** $rectGrid[len(rectGrid)-1].append(split(i))$
13. **if** $largestL = largestLforA$
14. **then for** $b \leftarrow 0$ **to** $hexCols-1$
15. **do** Calculate the interpolated values based on *isOdd*(b) and write value to *fileOut*
16. If original values are *noData* values, write value −1
17. $a \leftarrow a+1$
18. **if** $a > hexRows-1$
19. **then break**
20. **else** Write new-line character
21. $smallestLforA, largestLforA \leftarrow \lfloor \sqrt{3} \cdot a \rfloor, \lceil \sqrt{3}(a+\frac{3}{2}) \rceil$
22. **for** $p \leftarrow smallestL$ **to** $smallestLforA$
23. **do** $rectGrid.pop(0)$
24. $smallestL \leftarrow smallestLforA$

Algorithm INTERPOLATESQTOHEX translates a square grid to a hexagonal grid. However, this still does not give us any uncertainty range for a grid cell. Therefore, we need to compute uncertainty ranges. How we do this, is explained in the next chapter. For now, we assume that we have an input file with for each cell a low and a high elevation.

Since the data files are very large (several GB's), we need to take some memory management into account. One way of speeding up an algorithm for large data is by reading the data from a binary file.

In a binary file, the data is not stored as text, but as a sequence of bytes that are intended to be read as something other than text. One could see it as that the space in the file is divided into a number of records of fixed size and that each record represents a cell of the hexagonal grid. This is exactly how we want to use it. The file starts with three blocks containing the type "HEX", the number of rows and the number of columns, with a total size of b_1 bytes. For a grid of size $n \times m$, the file is followed by a sequence of $n \cdot m$ blocks – all having the same size b_2 – with the information of each hexagonal grid cell. When we want the information of the i'th hexagonal grid cell, we want to have the information in the file after $b_1 + (i-1) \cdot b_2$ bytes. In this way, we do not need to do a sequential read of the file, every time we want to access information of the file.

All the data manipulation so far writes the output into regular text files. When we want to use binary files, we need to convert our text files to this new binary format. To do this, we simply read the information from the text file and write it to a new file by creating blocks of information for each cell of the input. First, we create three blocks containing the type, number of rows and number of columns. Next, we read the low and high elevation values for a cell and create a block of bytes of this information and output this to the new binary file. After all cells are processed, we have a binary file containing the header and the cell information with the values as in the original text file.

6.2 Calculating with the uncertainty models

Given the binary files with an elevation range for each hexagonal grid cell, we can now compute the watersheds as explained in Chapter 4. Our implementation in C++ consists of a class implementing, for both flow models SFD and MFD, the three watershed algorithms. These functions are implementations of the algorithms described in Chapter 2 and 4 for calculating the potential, Q-avoiding potential, and persistent watershed of a set Q. Furthermore, we have two functions that *expand* a vertex, based on the flow type, and a function that calculates the *sdValue* of a vertex as explained in Section 2.5.

One enhancement in the code for calculating watersheds takes care of a large speed-up; for an input set S, the persistent watershed of a set $Q \subseteq S$ is defined as:

$$\mathcal{W}_{\sqcap}(Q) := \left(\mathcal{W}_{\cup}^{\setminus Q}((\mathcal{W}_{\cup}(Q))^c)\right)^c.$$

Per definition, $\mathcal{W}_{\cup}(Q) \subseteq \mathcal{W}_{\sqcap}(Q)$ and $(\mathcal{W}_{\cup}(Q))^c = S \setminus \mathcal{W}_{\cup}(Q)$. As one may recall, the Q-avoiding potential watershed of the potential watershed of Q is the set of nodes that can leave the potential watershed of Q in at least one of the realizations of the terrain. $N(\mathcal{W}_{\cup}(Q))$ is the set of vertices that are direct neighbours of the nodes in the potential watershed of Q and do not belong to the potential watershed themselves. It is trivial to see that the flow path of a node that leaves the potential watershed must include one of the vertices in $N(\mathcal{W}_{\cup}(Q))$. That means that, instead of taking the complement of $\mathcal{W}_{\cup}(Q)$ to calculate the Q-avoiding potential watershed, we can simply take the set $N(\mathcal{W}_{\cup}(Q))$ to calculate it for. Enqueueing all these neighbours at the lowest possible elevation for each node, when running the algorithm, results in the same Q-avoiding potential watershed of the complement of the potential watershed of Q, as when one would calculate the Q-avoiding potential watershed for the set of $S \setminus \mathcal{W}_{\cup}(Q)$. In this way, we do not need to check all the other nodes $(S \setminus \mathcal{W}_{\cup}(Q)) \setminus N(\mathcal{W}_{\cup}(Q))$, saving an enormous amount of time for large sets $(\mathcal{W}_{\cup}(Q))^c$. The adapted algorithms are described by Q-AVOIDINGPOTWSADAPTED and PERSISTENTWSADAPTED.

The results of the watershed algorithms (which nodes are part of which watershed and on which lowest elevation a node was enqueued) are then captured in a PNG file. The results of the algorithms are discussed in the next chapters.

Algorithm Q-AVOIDINGPOTWSADAPTED(S, Q, P)
Output: nodes $p \in P$ that can leave P by flowing to a node in S
1. For all $s \in S$: Enqueue (s, z) with key $z = low(s)$
2. **while** the Queue is not empty
3. **do** (s', z') = DequeueMin()
4. **if** s' is not already in the output set $\wedge s' \notin Q$
5. **then** Output s' with elevation z' if $s' \in P$
6. Enqueue each $(p, z) \in$ EXPAND(s', z') if $p \in P$

Algorithm PERSISTENTWSADAPTED(Q)
1. $\mathcal{P} \leftarrow$ POTENTIALWS(Q)
2. $N(\mathcal{P}) \leftarrow \{n : n$ neighbour of a node in $\mathcal{P} \wedge n \notin \mathcal{P}\}$
3. $\mathcal{A} \leftarrow$ Q-AVOIDINGPOTWSADAPTED$(N(\mathcal{P}), Q, \mathcal{P})$
4. Output $\mathcal{P} \backslash \mathcal{A}$

CHAPTER 7

Computing the Elevation Range

In this chapter we discuss several methods to determine the elevation range of a node. As input data set, we use the DEM of the Neuse River Basin in North Carolina to test our settings. Note that this means that we introduce uncertainty on a model for which preprocessing has occurred to extract the data points from original data, for instance, a LiDAR data set. For future work, it would be interesting to see the results of the methods applied to this original data, including all noise and uncertainty, taking the original uncertainties into account and get different uncertainty ranges as a result. For now, we apply the methods on the DEM as it is. The hexagonal grid file of this data set — without uncertainty ranges — is drawn in Figure 7.1, together with the colour maps that are used for visualizing the elevation data of the terrain, the minimal elevation for which a cell belongs to the potential watershed of a set of cells and the colour for the low elevation of a cell in the persistent watershed.

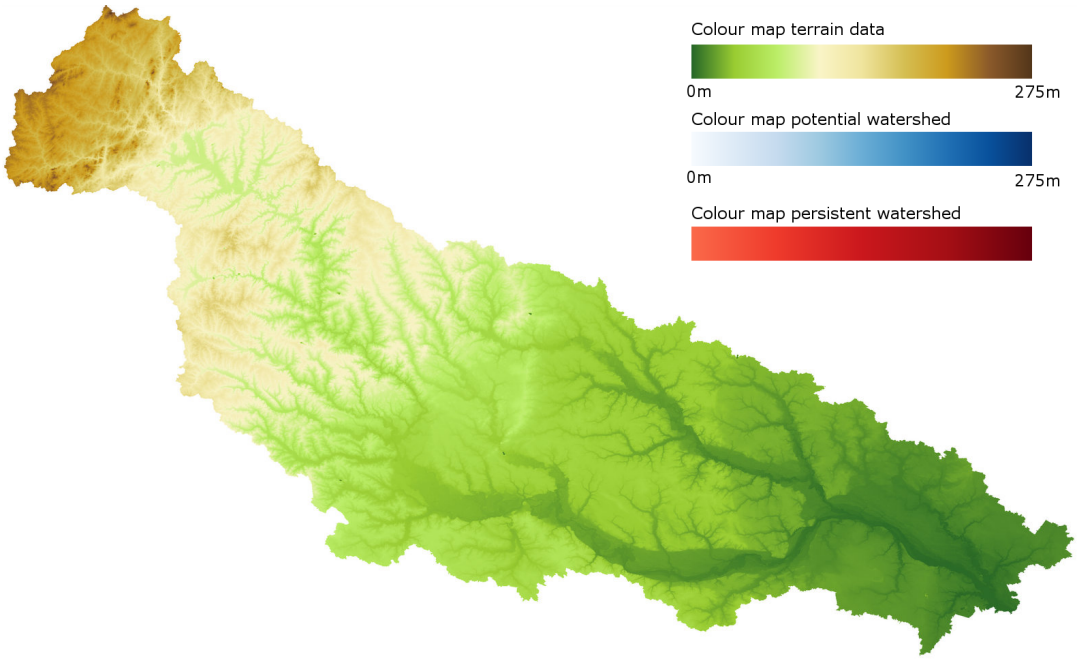
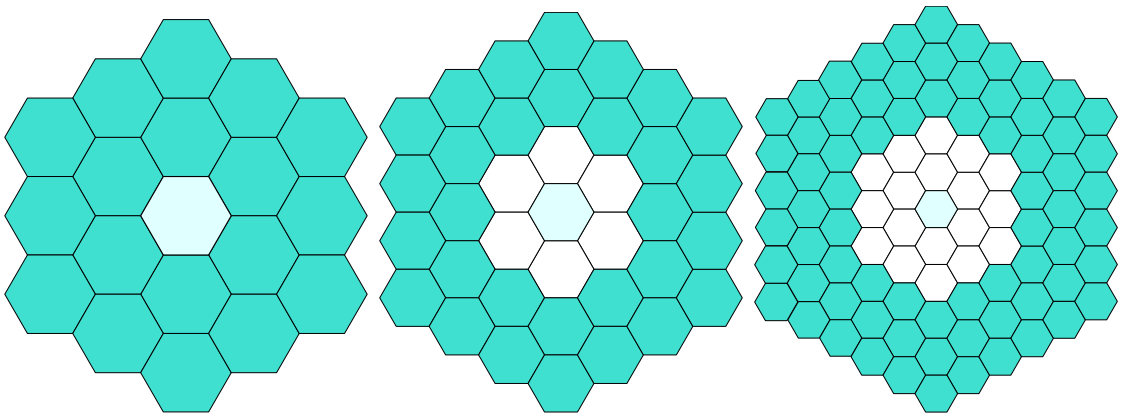


Figure 7.1: Data set of the Neuse River Basin in North Carolina (215km by 153km wide), interpolated to hexagonal grid data file, together with the colour maps used in the rest of this report.

In the next paragraphs, we calculate the watersheds for cell 41924617; this cell has the largest watershed in the original data set. For simplicity, we will refer to this cell number as cell c_w.

7.1 Locality setting

All interpolation methods described in this chapter are executed using three different locality settings; interpolating and determining the ranges is done by looking at the elevation values of neighbour nodes. We have chosen three different neighbourhoods to calculate the ranges with, are depicted in Figure 7.2.



(a) Setting 18 **(b)** Setting 30 **(c)** Setting 72

Figure 7.2: The three settings for determining the uncertainty range of a cell. Each setting shows which neighbours are involved during the calculations for the interval of the light cyan cell; white cells are ignored and the information of the turquoise cells is used for determining the interval.

The first setting, Setting 18, looks at the direct neighbours within two rings distance. The second setting, Setting 30, does not look at its direct neighbours, but at the two rings around them, and the last setting, Setting 72, only looks at the neighbours in ring 3,4 and 5. Using the information of these neighbours we try to find a plane or a polynomial surface that could resemble the area around the current cell in the actual terrain. With this fit, we give an estimation of the possible elevation of the current cell. We then assume that the real elevation of the cell will lie between the predicted elevation and the measured elevation. The reason why we chose these different approaches is based on the appearance of local errors. Based on the sampling rate one would want to look a bit 'further' away in the area to estimate the surface, since it could be possible that the current cell lies in, for instance, a local minimum.

Since one of the main preprocessing steps for hydrological analysis in GIS is to remove supposedly artificial local depressions (local minima), ideally, the uncertainty intervals would allow the water of a node in a local minimum to leave this depression in a way. Therefore, our first method, the min-max method, was based on taking the minimum and maximum elevation of a number of neighbouring nodes. However, the watersheds computed for points in the terrains created by this interpolation method, are unrealistic compared to the original terrain. The idea of the min-max method for a cell c is to look at a neighbourhood setting, say for example Setting 18, and set as low elevation the minimum elevation value of the current cell and the 18 neighbours and as high elevation the maximum of all possible elevations of the current cell and the 18 neighbours. In Figure 7.5, the watersheds are calculated for cell 41924617, which lies within the indicated red circle. As one would expect, water will only flow to that cell from a number of the cells in the same valley the cell belongs to. However,

the Figure shows otherwise. In this Figure, it seems that all cells with a height that is higher than cell c_w have a possible flow path to this cell. In reality, this can not be true. We can explain these erroneous results by looking at the way in which the watersheds are calculated. A neighbouring cell $n(c)$ of cell c is added to the potential watershed if the elevation of $n(c)$ can be set such that c is the steepest descent neighbour. If the minimum and maximum elevation of a cell depend on the minimum and maximum possible elevations of a number of neighbours and the current cell, it is possible to build a sort of dam in almost every situation and cells that normally belong to a different valley, now can set their elevations to the same elevation as a surrounding neighbour, enabling a water flow to a different valley. In Figure 7.3, we have sketched an example of a 1.5 dimensional elevation-bounded terrain. The thick lines describe the upper and lower terrain after applying the min-max method to the original terrain (indicated with the normal black line). Point b in the original terrain will always flow to the right valley, via point c. However, after applying the min-max method using Setting 18, point c and d can be set to the same elevation as b's elevation in the original terrain, allowing water to flow to the valley on the left of b, via point a. By lifting the points to one of the original elevations of the neighbours in the locality setting, the steepest neighbour of a point p, can now be any of the six neighbours of p.

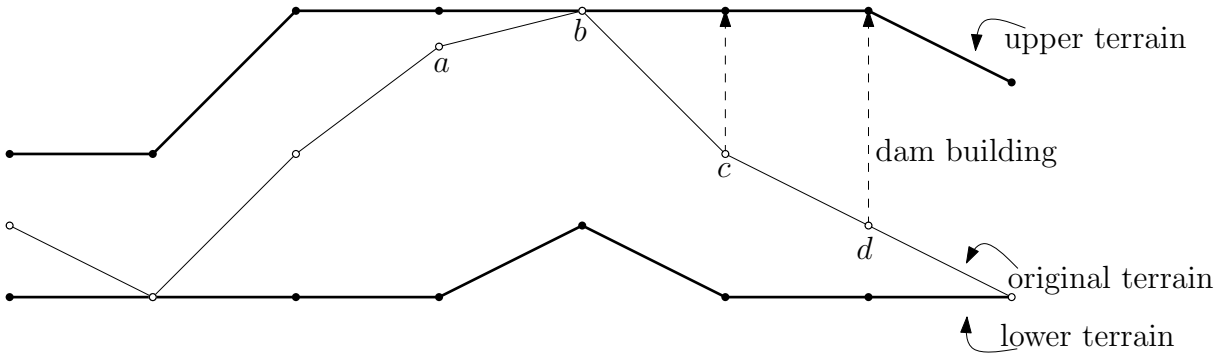
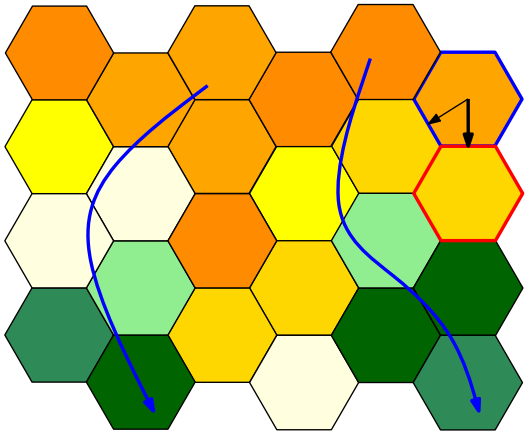


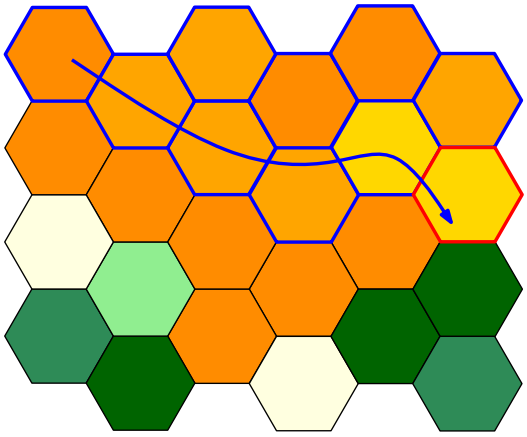
Figure 7.3: An example of a 1.5 dimensional imprecise terrain, with the possibility to dam-building, after applying the min-max method Setting 18 to the original terrain.

In Figure 7.4, we have sketched the situation in a hexagonal grid terrain. Figure 7.4a shows the elevations of the cells in the orginal terrain, and Figure 7.4b shows a possible realization of the elevation-bounded terrain that is obtained after applying the min-max method to the original terrain. In the original terrain, we can distinguish two valleys, one valley following the flow path indicated by the left blue arrow, and the other valley indicated by the right arrow. These valleys are isolated by the ridge of the orange cells in between. However, after applying the min-max method with Setting 18, we can find a realization in which water from the left valley can flow to the right valley, by building a dam. Let us denote the cells from the left valley that are in the potential watershed of the red cell by V. Each cell $v \in V$ has a neighbour $n(v)$ that is higher (or equally) situated and delivers water to v on its way to the red cell. Because of the min-max method, it is possible for the lower cells $l \in L$ of the left valley, where the water from points $v \in V$ normally would flow to, to set their elevations to the elevations of the water delivering neighbour cells $n(v)$. This blocks the water flow downwards the valley and enables a water flow to neighbours with the same elevation, causing a sort of 'isoline-flow-path', which is unrealistic in the original terrain.

This means that basing the uncertainty range on the minimum and maximum elevation of the direct neighbourhood does not give us the result as we would like. Therefore, we dived into the method of Ordinary Least Squares, as described in the next section.



(a) Original terrain and its general flow directions.



(b) Possible realization of the elevation-bounded terrain after applying the min-max method with Setting 18.

(c) Colormap for the terrains in (a) and (b).

Figure 7.4: An example of dam-building on a hexagonal grid terrain. The cells in the potential watershed of the red outlined cell have a blue outline. The blue arrows indicate the flow on the terrain.

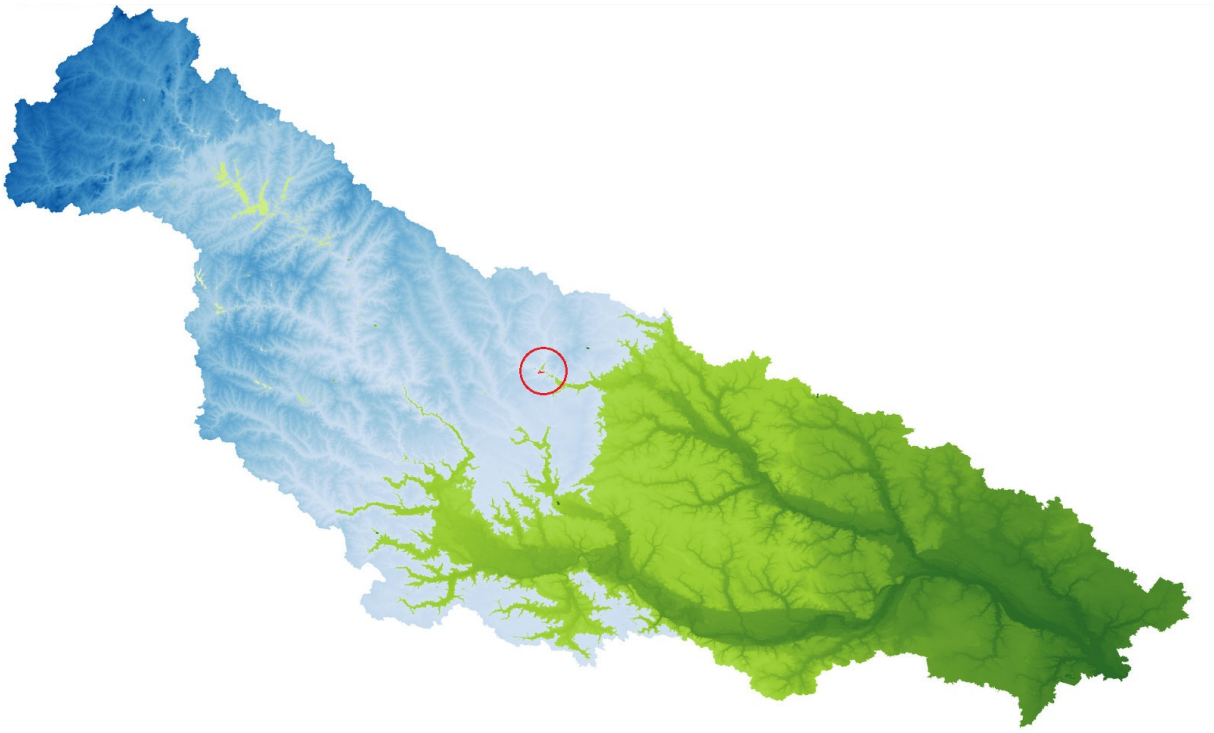


Figure 7.5: Computed watersheds for cell 41924617 within the indicated red circle, based on the min-max method.

7.2 Ordinary Least Squares

One of the causes of computational uncertainties are the local-scale errors that appear in the data sets. These local scale errors, can have large effects on the result of an algorithm. Therefore, we thought that smoothing the data, while keeping the original elevations in mind, could help us by obtaining better results. One way of smoothing the data is by trying to fit a certain shape onto a specific area to decrease the number of outliers and to increase the consistency between all data points. We will first explain which fitting method we selected. After the introduction to this method, we explain how we apply the method in our situation.

Ordinary Least Squares (OLS) is a method for estimating a fit for a linear regression model by minimizing the sum of squared (vertical) distances between the values of the data file and the values of the linear approximation. We use OLS on independent input values: x and y, which represent the coordinates of a hexagonal grid cell centre, and z, the elevation value of a cell. On page 108 Davies [Dav08] describes the general case of the linear regression OLS method. OLS tries to fit a certain model on a number of observations. We will explain the OLS method now for fitting a plane $z = a \cdot x + b \cdot y + d$ through a set of points, in other words, finding a suitable a,b and d for which the equation returns values that are, on average, as close as possible to the original values. For each observation (for example, a z-value of the data file that belongs to a cell c on position (x, y)), the x and y values are added to the *design matrix* X, and the z values are added to the observation matrix Y. This observation matrix contains only one column, with on each row i the elevation value of a point corresponding to the data of the point on row i in the design matrix. The design matrix has three columns $[1, x, y]$. The rows of the design matrix contain the values of the independent variables x and y. An all-one column is usually added to be able to predict the constant term d. Leaving the all-one column out would result in fitting a plane $z = a \cdot x + b \cdot y$. If variables occur in different forms (x, x^2,etc.), the value of x, respectively x^2, is added to the matrix. For more information on OLS and other related procedures, we refer to Montgomery [Mon09].

Let us given an example. We want to fit our plane $z = a \cdot x + b \cdot y + d$.
For the i'th point $(x_i, y_i, z_i) = (2,4,7)$, we have in the observation matrix the value 7 on row i, and $[1, x_i, y_i] = [1,2,4]$ on row i in the design matrix. However, if we try to fit a polynomial surface such as $z = a \cdot x^2 + b \cdot xy + d \cdot y^2 + e$, we have for this same point a value of 7 in the observation matrix, and $[1, x_i, x_i \cdot y_i, y_i^2] = [1,4,8,16]$ on row i in the design matrix.

The fit is determined by minimizing the least squares estimator. For each of the n observation values we have that $Y_i = \beta_1 X_{i1} + \beta_2 X_{i2} + \ldots + \beta_k X_{ik} + \epsilon_i$ where $1 \leq i \leq n$ and ϵ_i is an independent random variable, representing the noise in both the observation and measurement [Dav08]. For OLS, one typically assumes Gaussian white noise. If each distortion has a normal distribution with zero mean, the signal is said to be Gaussian white noise. In this case, OLS is the maximum likelihood estimator for the unknown parameters. When the errors are non-Gaussian, the estimates may be biased or grossly inaccurate and corresponding statistics, such as p-values, are no longer valid. We try to minimize the sum of squared vertical distances between the values of the observation values (for k columns in the design matrix):

$$\min_{\beta} \sum_{i=1}^{n} \left(Y_i - \sum_{j=1}^{k} X_{ij} \beta_j \right)^2 = ||Y - X\beta||^2 = (Y - X\beta)^T (Y - X\beta),$$

which results in

$$\beta = (X^T X)^{-1} X^T Y.$$

The inverse of matrix $(X^T X)$ only exists if the colums of design matrix X are independent (which is the case in our situation, as explained further on in this section). OLS calculates $\beta = (X^T \cdot X)^{-1} \cdot X^T \cdot Y$, resulting in a vector with the number of elements equivalent to the number of columns in the

design matrix. In our plane fitting case, these are three elements. The plane that we are searching for can now be described by $z = \beta_1 \cdot x + \beta_2 \cdot y + \beta_3$.

Using the statsmodels package for Python, we can implement this method very easily by selecting the locality setting (which neighbours have influence on the fit) and computing the fit for these cells. As a result, we get as output the values for the vector β, together with the coefficient of determination for the corresponding fit, the *R-squared* value. This value indicates the goodness-of-fit of the regression; it will be equal to one if the fit is perfect and it will be equal to zero when the so-called entries X in the design matrix have no explanatory power regarding the observations Y. We also have the *R-squared Adjusted* statistic, which is designed to penalize for the excess number of regressors which do not add to the explanatory power of the regression. This value is always smaller than the *R-squared* value and is defined as follows:

$$R^2_{Adj} = 1 - \frac{n-1}{n-p}(1-R^2).$$

The x and y values referring to our grid coordinates are clearly independent. Furthermore, we can transfer all uncertainty in these x and y values to the z variable. We then have known values for x and y and a measured value for z. Now we can apply OLS with the x, y and z to spread out the uncertainty in z over a larger area, such that the noise on the fitted elevation z' decreases and we can find a better estimate for the final z value in a specific point (x, y) by using the x, y, z values of surrounding points to calculate a fit.

We want to calculate the uncertainty range for cell c. We apply this OLS method in three different ways:

1. The first application (OLS Plane) tries to fit a plane ($z = a \cdot x + b \cdot y + d$) through a set of points. An OLS fit then returns fitted values for a, b and d.

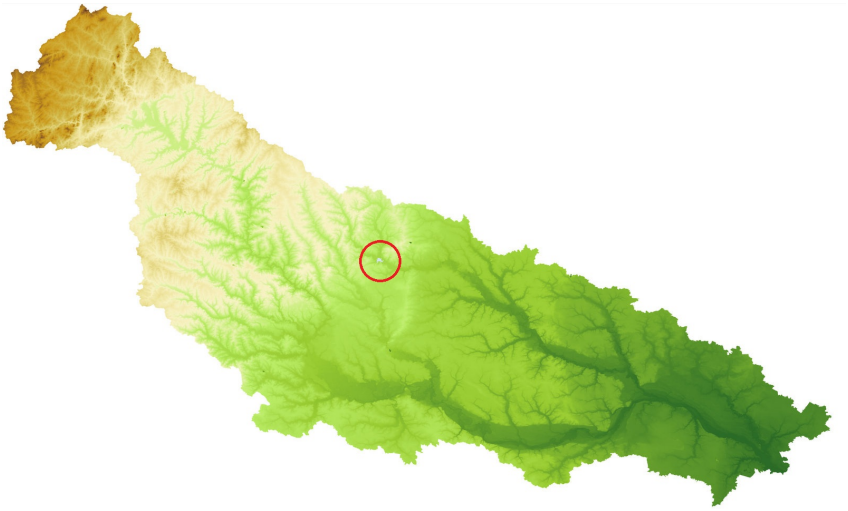
2. The second application (OLS PlaneSurf) tries to fit a plane as described in **1.** and a polynomial surface ($z = a \cdot x^2 + b \cdot xy + d \cdot y^2 + e$) and selects the fit with the best *rsquared* value.

3. The last application (OLS PlaneSurf Adj.) again tries to fit a plane and a polynomial surface, both as described above, but bases its choice of the best fit on the *adjusted rsquared* value.

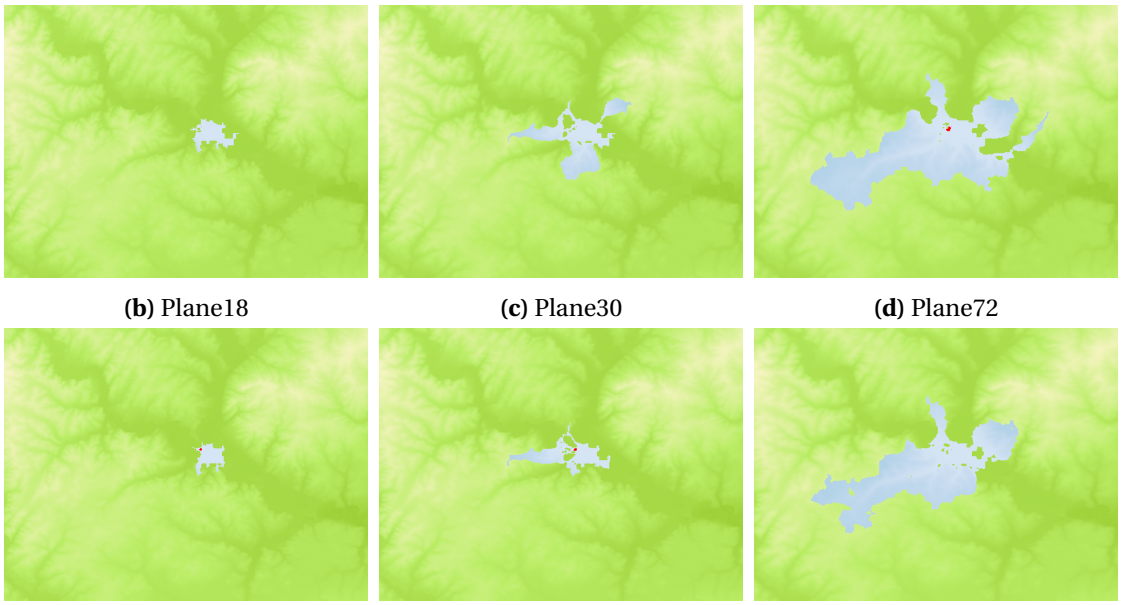
Based on the selected setting, the point set consists of the 18, 30 or 72 neighbours as depicted in Figure 7.2. Note that it is impossible to find a fit of a plane when the observation points are linearly dependent – for example when all points are placed on one line – because that gives you information on only two of the three degrees of freedom. However, in our situation, we are looking at neighbours in all possible directions, resulting in most cases in a set of observation points that are not linearly dependent. If there are too few neighbours, or the set is linearly dependent (when the neighbours in a setting of one specific 'line' are only useful for calculations), the original z value is used. Otherwise, the fitted z value for a cell is used to update the minimum and maximum value of the cell. If the fitted value is larger than the current value, the maximum value is updated to the fitted value and the minimum value to the original value. If the fitted value is smaller than the current value, the minimum value is updated to the fitted value and the maximum value to the original value.

In Figure 7.6 the results are drawn for the OLS Plane and OLS PlaneSurf methods. The six pictures are captured and enlarged from the area with the red circle around it. A hexagonal grid cell in this data set has a diameter of roughly 18.3 meter.

We can see that the water indeed flows along the mountain hill down into the river area, but water from the higher parts of the river does not end up in the area, as opposed to our expectations. This phenomenon can be explained by looking at the interpolation methods we use to establish the uncertainty interval. In the case that a local minimum is as big as the neighbourhoods we are looking at, the resulting surface will also be approximated as a local minimum. Therefore, flow entering this local minimum is still not able to leave this local minimum, and the watershed algorithm will not



(a) Area captured and zoomed below for specific approaches



(b) Plane18 **(c)** Plane30 **(d)** Plane72

(e) PlaneSurf18 **(f)** PlaneSurf30 **(g)** PlaneSurf72

Figure 7.6: Results of the different interpolation methods on the Neuse data set.

capture these nodes in the potential watershed of points somewhere else on the terrain. Therefore, for an actual flow estimation, dealing with uncertainty in a node is not enough. In an area where a high number of extrema appear on a river path, water will (eventually) flow to the river estuary, but not when we consider the actual surface of the terrain. The water will only flow once the river contains a certain amount of water and the local minima are flooded. Therefore, we still need to take a look at *flooding*, as is done on regular GIS DEM's.

7.3 Flooding

Flooding calculates the height of the water level of each cell after an infinite amount of rain has fallen on the terrain. In this way, pools and lakes are formed on parts of the area where water could not immediately flow downwards to a lower situated part of the terrain, or by leaving the terrain into the

so-called *ocean*, the area surrounding the original terrain. After a certain amount of rain, enough water has fallen to fill all local minima and form lakes such that the next rain that falls is able to flow off the terrain, into the ocean. The moment that these local minima are filled and water falling down at each cell is able to flow into the ocean, we call the terrain *flooded*. The height up to which a cell is flooded, is the minimal height of a flow path from that cell to an ocean cell. The height of a flow path is the highest height of a cell occurring on that flow path. A way to calculate this height is by turning the definition around and by taking a look at the definition from the ocean's point of view. Based on that setting, we want to find the Single Source Lowest Path (SSLP) [Jan08] from the ocean to the cell we are interested in, which is quite similar to the well known Single Source Shortest Path (SSSP) problem. As Janssen explained, the only difference is the way the value of a path is calculated; for SSSP the sum of the weights of all edges on a path is calculated, where for SSLP only the maximum weight, is important. This weight is related to the height of the lowest path between the two cells it connects.

We used Janssen's [Jan08] implementation for flooding a regular squared grid and adapted the code – the I/O-efficient implementation with the cache-oblivious approach, see Secion 2.3 of his report [Jan08] – to the case of a hexagonal grid input. In this way, we are able to calculate the minimum height for a cell such that water is able to flow off the terrain. However, since we deal with uncertainty ranges rather than with a fixed elevation value for a grid cell, we had to make several other decisions. For a terrain with cells c with elevation range $[c_l, c_h]$, we flood the area based on the minimum elevation values c_l and as a result we get the new elevation values c_f for each cell such that water can leave the terrain and flow into the ocean. Next, we update all ranges for the cells where $c_f > c_l$ such that the new range becomes $[c_f, c_f + (c_h - c_l)]$. In this way, we keep the original uncertainty of a cell in mind when flooding the terrain. No-data cells are ignored by the implementation if they have a path consisting of only no-data cells connecting the cells to the ocean. If such a path does not exist, the cells are treated as regular cells and they are flooded as well. 'Holes' in a data set are thus removed. In the next chapter, we discuss the results of the watershed algorithms on the flooded data sets.

CHAPTER 8

Evaluation

Since determining whether water can flow from one point to another point in the slope-bounded model is proven to be NP-hard (Chapter 5), the actual computations we evaluate in this chapter, are done on elevation-bounded models. In this chapter, we show some of the results of the algorithms on the different data sets as created according to Chapter 7. Based on these results, we reflect on the existing algorithms. We end this report with some suggestions for further research.

8.1 Results

All images in this and previous chapters show the minimum elevation values for the terrain data, unless indicated otherwise. Furthermore, the computations are done for the SFD flow method, where water flows to the steepest descent neighbour, unless indicated otherwise. The third method of interpolation to find the uncertainty range for a cell, OLS PlaneSurf Adj., gave similar results as the second method, OLS PlaneSurf. Since this did not contribute to the conclusions we draw, we did not include these results in this report. In appendix A, extra results are given, contributing the statements made in this chapter.

8.1.1 Certainty of watersheds

Since the difference between the size of the potential watershed and the size of a persistent watershed tells us something about the certainty of a watershed, we could think of a map showing the certainties for each set Q for which the watersheds are calculated. We define the certainty of the watershed of a set Q as:

$$\text{certainty}(Q) = \frac{\text{size(persistent watershed)}}{\text{size(potential watershed)}}.$$

Due to the fact that $\mathcal{W}_{\cap}(Q) \subseteq \mathcal{W}_{\cup}(Q)$, we know that if the size of a persistent watershed of Q is equal to the size of the potential watershed of Q, all water arriving at a cell that lies in the potential watershed of Q will eventually flow through at least one of the cells in Q. In other words, because the potential watershed is equal to the persistent watershed of Q, it is certain for every cell that it belongs to the watershed of Q. If there is a big difference in size between the size of the potential watershed of Q and the size of the persistent watershed of Q, it is for a large number of cells not certain whether water arriving at that cell will eventually reach a cell in Q. Therefore, the watershed is more uncertain in these cases. Based on this observation, we compute the certainty of a watershed and label ten

different categories, as shown in Table 8.1. For example, category 4 depicts the watersheds that have a certainty between 0.4 and 0.5, where 0.4 belongs to the range when counting, and 0.5 does not.

We calculated the watersheds for blocks of 10x10 cells (roughly 137x137 m^2), 50x50 cells (roughly 685x685 m^2) and 100x100 cells (roughly 1.37x1.37 km^2) for a part of the Neuse basin as depicted in Figure 8.1. We used the colour for these plots as depicted in Figure 8.2; a block is coloured darker red if the watershed of that block has a high uncertainty, whereas the block is coloured lighter yellow when the potential and persistent watershed are almost equal in size. We plotted the uncertainty results for each of the blocks, as shown in Figure 8.3. Together with these calculations, we kept track of the number of blocks in each of the ten categories as mentioned above. In Figure 8.4, Figure 8.5 and Figure 8.6 these results are plotted in bar charts, for a better overview of the exact partition of the different data sets and to detect a change in uncertainties for larger block size.

0	[0-0.1)
1	[0.1-0.2)
2	[0.2-0.3)
3	[0.3-0.4)
4	[0.4-0.5)
5	[0.5-0.6)
6	[0.6-0.7)
7	[0.7-0.8)
8	[0.8-0.9)
9	[0.9-1]

Table 8.1: Certainty categories

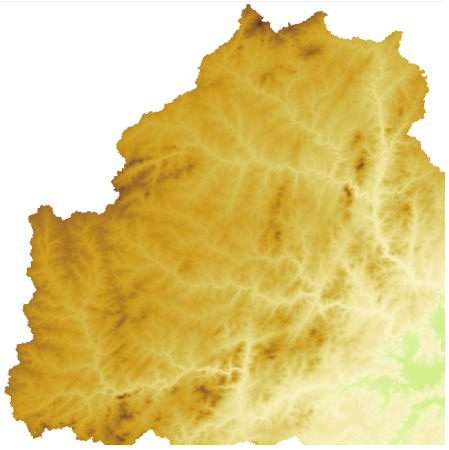
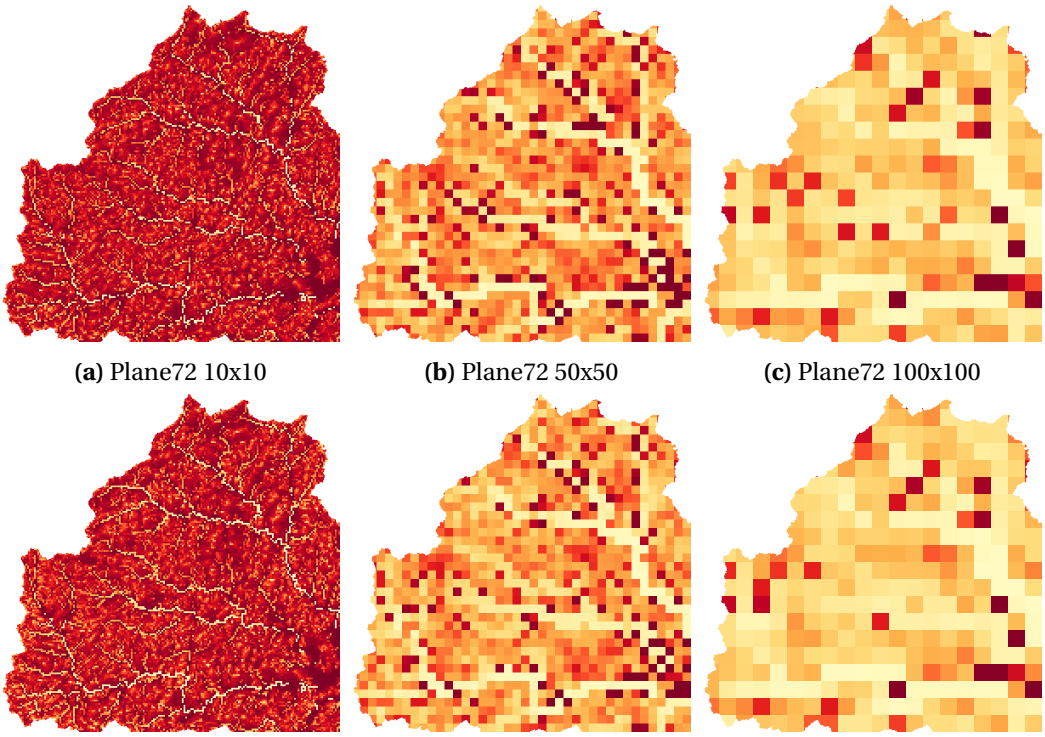


Figure 8.1: Area (36.5x36.5 km^2) for calculating certainty

Figure 8.2: Colour map for certainty (range [0-1])

At first sight, the figures of the Plane72 method seem to show the same results as the figures for the PlaneSurf72 methods. But when we look at the bar chart in Figure 8.6, we see a small difference. For the second method, the bars show a higher percentage for the higher certainty categories, compared to the first method. That means that smoothing the area, while keeping the original terrain in mind, based on a fit of both a plane and a polynomial surface gives slightly better results for the computation of watersheds on the terrain. A very clear difference, when looking at Figure 8.3, is that for larger block size, the uncertainty of the watersheds decreases. This is confirmed by the bar charts in Figure 8.5 and Figure 8.6; for the 10x10 blocks more than 50% of the computed watersheds end up in the first category (number 0), which means that for each of these computations the potential watershed is more than ten times larger than the persistent watershed of the block. For the 50x50 blocks we see a higher concentration around the center categories (4,5 and 6) and for the 100x100 blocks the majority of the computations is categorized in category 6 or higher.

We did some experiments on the data to find out how this could be explained. We selected a slightly smaller area of 11 km high and 12.4 km wide. In Figure 8.7, the watersheds are calculated for three different block sizes, all starting from the same cell in a sub area from the area shown in Figure 8.1. The blocks for which we calculate the watersheds are coloured firebrick red. We see that the cells in the potential watershed of the 10x10 block in Figure 8.7a almost all belong to the potential watershed and only a few cells apart from the block itself belong to the persistent watershed of the



(a) Plane72 10x10 **(b)** Plane72 50x50 **(c)** Plane72 100x100

(d) PlaneSurf72 10x10 **(e)** PlaneSurf72 50x50 **(f)** PlaneSurf72 100x100

Figure 8.3: Results of the different interpolation methods on the Neuse data set.

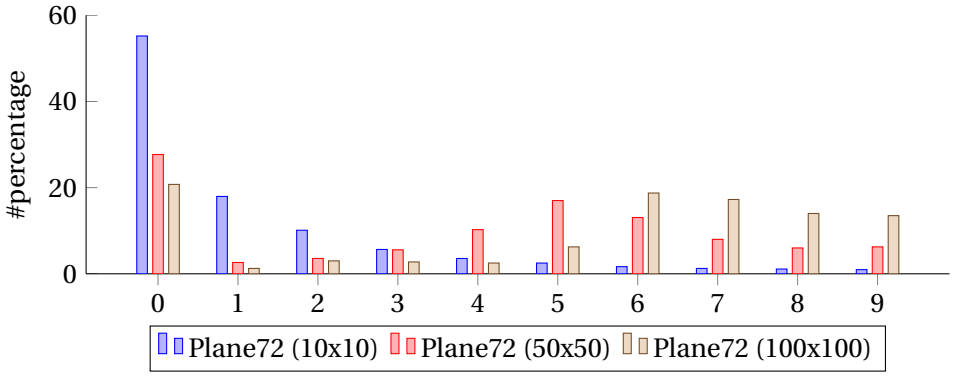


Figure 8.4: Uncertainties in the Plane72 flooded data file, for three different block settings.

block. When we look at the two other figures, Figure 8.7b and Figure 8.7c, both for larger block size, we see that the persistent watershed suddenly is quite a lot larger compared to the smaller 10x10 block size watershed. When we calculate the watersheds for a 10x10 block based on a cell a little further upwards on the river, where the 10x10 block covers the width of the 'river' – see Figure 8.8, the persistent watershed is all of a sudden very large again. In the next section, we try to explain why this is the case and think of methods how to prevent these situations.

Furthermore, we can recognize the original area as depicted in Figure 8.1, which implies that the type of area has influence on the results of uncertainty. For blocks in and near the 'rivers' in the valleys, the watersheds are more certain than on higher grounds. Blocks that are close to the 'rivers' have a very high uncertainty. The reason for this is explicable by the same explanation as the explanation for the above scenario, which is given in the next section. Blocks that are positioned on higher ground have a high uncertainty, but when we realize what we calculate in these situations,

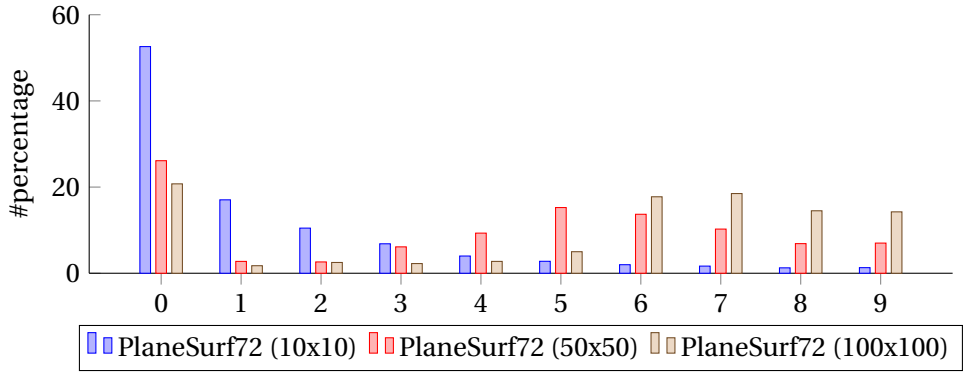


Figure 8.5: Uncertainties in the PlaneSurf72 flooded data file, for three different block settings.

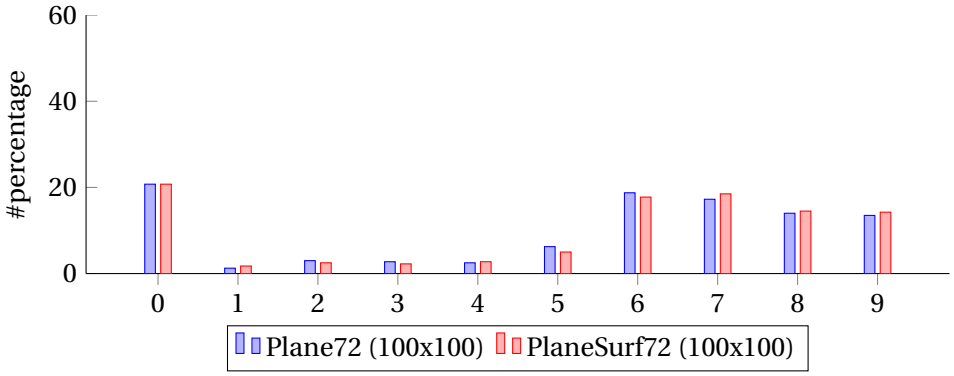
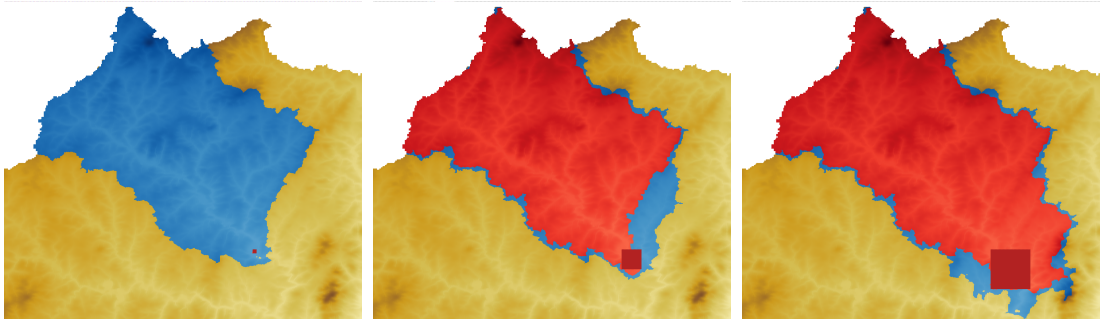


Figure 8.6: Uncertainties of the 100x100 block setting for Plane72 and PlaneSurf72 compared.

this makes sense. The certainty of a watershed says something about the proportion of the potential watershed compared to the persistent watershed. In the case of cells that are on higher ground, the potential watersheds are quite small, and the persistent watershed is also very small. Due to the fact that these areas suffer from local roughness, it is very likely that, although the watershed is very small, a number of these cells in the watershed can avoid flowing throug a cell in Q. Therefore, a large part of the potential watershed is not in the persistent watershed. Because the watershed of such a cell is very small, an extra cell more or less in the potential watershed suddenly gives a large difference in certainty. Take for example a persistent watershed of size 1. When the potential watershed has size 995 or 100, this both results in a certainty of roughly 0.001. When the potential watershed has size 5 or 10, this gives two different certainties 0.2 and 0.1. This immediately clarifies the fact that parts of the 'rivers' in the valley have a higher certainty; the size of the watersheds for cells down in the valley are very large, because all water from the cells uphill end up in the river some time. That means that a few cells more or less in the potential watershed have less influence on the certainty of the watershed. This also explains the observation that for uphill areas, the uncertainty decreases rapidly for higher block size. Since the watersheds in the uphill areas are smaller than for the valley 'rivers', taking a larger block size increases the size of the persistent watershed, but the size of the potential watershed increases by only a number of cells, as depicted in Figure 8.9.

Another observation we can make, is that the certainty of the watersheds increases for the lower locality settings (interpolating based on a smaller neighbourhood), see A.3. This makes sense, since taking more measurements into account from areas further away, will influence the fit of a plane; for a larger neighbourhood, the uncertainty ranges become slightly larger. As we saw with the min-max method, if ranges become too large, the dam-building problem appears again, but now on a more local scale.



(a) PlaneSurf72 10x10 **(b)** PlaneSurf72 50x50 **(c)** PlaneSurf72 100x100

Figure 8.7: Uncertainty decreases for larger blocksizes (area is 12.4km by 11km)

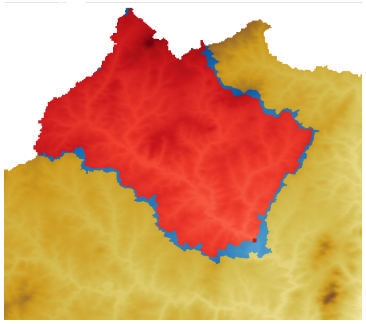
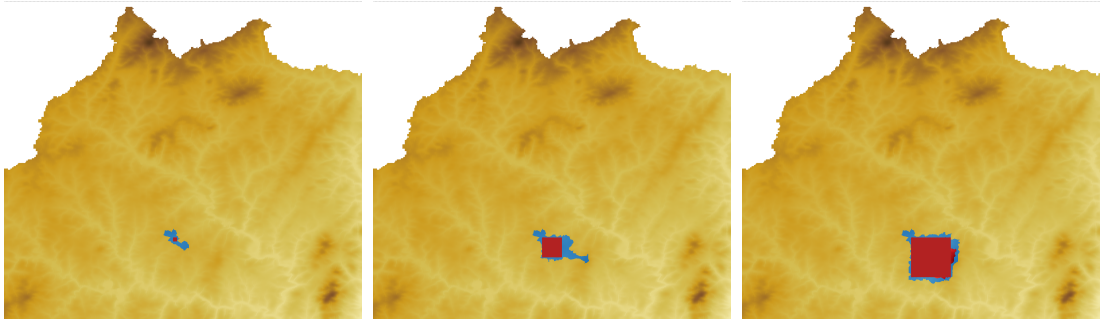


Figure 8.8: Watersheds of a 10x10 block a little further upwards the river



(a) PlaneSurf72 10x10 **(b)** PlaneSurf72 50x50 **(c)** PlaneSurf72 100x100

Figure 8.9: Uphill watersheds stay small, resulting in less uncertainty for larger block sizes.

We also calculated the certainties for the case of the MFD method, where water flows to all neighbours that have a lower or equal elevation as the cell we are looking at. The results are similar, see Appendix A for the Plane72 results – for larger block size the uncertainty decreases, and for the PlaneSurf methods the results give slighlty better results than the Plane methods. However, in general the uncertainty of the blocks is higher, which means that there is a larger difference between the size of the potential watersheds and persistent watersheds. This can be explained by the fact that cells are added to the potential watershed if there is a setting such that at least one of the neighbours can be positioned at a lower height and water from this cell can flow to a cell in Q. But for these cells there are also other settings where they will not have a higher height and water will flow to other cells that will never reach a cell in Q. As a result, the potential watershed calculated by using the MFD method is usually larger than the potential watershed of a set of cells for the SFD method, containing more cells that are also contained in the Q-avoiding potential watershed.

8.2 Evaluation of the watershed algorithms

During our computations we stumbled upon a number of problems. Below, we will discuss each of the problems and, where applicable, suggest solutions for the problems.

8.2.1 Single-cell behaviour

One of the main drawbacks of the watershed algorithms is the fact that the calculations are done in a single-cell fashion, meaning that a tiny change in the input set Q can have a huge effect on the resulting watersheds. One would expect that adding one cell more or less from the same area to the set Q does not have such a great impact on the results. This turned out not to be true. Especially in the case of an area where cells have a similar height, such as a flooded terrain, the algorithms return confusing results and you need to understand the behaviour of the watershed algorithms very well to draw any conclusions on, for instance, the certainty of the watershed.

The certainty of a watershed — size(persistent watershed) versus size(potential watershed) — gives an idea of the amount of water that could flow via the set Q. In the case that the persistent watershed is very small compared to the potential watershed, one would intuitively assume that there is a low probability that all the water from the potential watershed will eventually flow through Q. However, this is in most cases not the case. The reason why there is a large difference in size in many cases is causes by the fact that water can avoid the set of cells via a number of neighbouring cells that do not belong to the set Q.

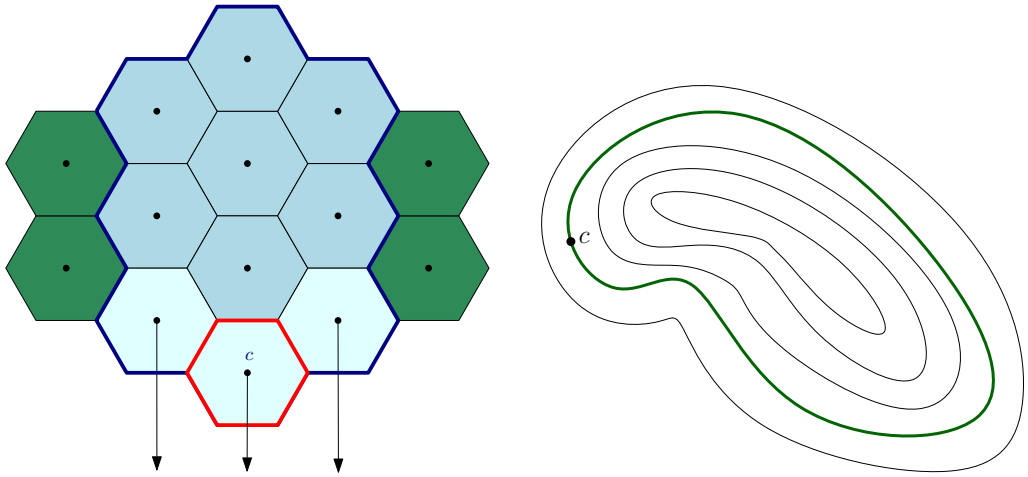
In Figure 8.10a, a situation is sketched for which the algorithms returns a confusing answer. In this example — where only a part of the terrain is drawn for simplicity — the persistent and potential watershed of cell c are depicted; the red outlined cell, cell c, is the only cell in the persistent watershed, all blue cells together with cell c belong to the potential watershed. The darker blue cells have a higher elevation than the light blue cells and the green cells have the highest elevation. The arrows leaving from the light blue cells indicate that water from these cells will always flow downwards to their neighbouring cells. According to the watershed definitions, water from the darker blue cells may avoid flowing through cell c on their path to the ocean. This is indeed the case, since water can flow around cell c via the other two light blue cells, where it gathers again afterwards to flow further downwards. However, this is a very unrealistic situation, and we are not interested in the behaviour of water avoiding single cells, but in the general behaviour of water in that area. When we would have selected the three light blue cells as a set to find the potential and persistent watershed, all dark blue cells would have been part of the persistent watershed as well.

With this single-cell behaviour formulated, we can explain the results of Figure 8.3; The blocks that cover the lowest cells in the 'rivers' of the valleys have a high certainty, since water can not 'escape' flowing through these lowest cells. Blocks that contain a number of lowest cells of the 'river', but not all, calculate the potential watershed and end up with the cells that will flow through these lowest cells, but can avoid these cells by flowing through the neighbouring cells of the river, that are not included in the block. This results in a small persistent watershed, since a large part of the uphill cells can avoid the cells in Q by flowing via the neighbouring river-cells. On higher ground, the watersheds are smaller, but the difference of a small amount has a great impact on the uncertainty of a watershed, causing a lot of blocks ending up in the low-certainty categories.

Possible solution and drawback

We thought of a solution for this problem, by augmenting the set Q with all direct neighbours with the low elevation c_l the same as the lowest elevation for a cell in Q. Although this would lead to better (more certain) results — results where the persistent watershed have a size in the order of the size of the potential watershed — for a large number of calculations, the resulting watersheds are not related to the (actual) watersheds of the original set Q.

For cells Q' in a flooded area, the new algorithm would add all neighbouring cells within that are flooded as well to the set Q of the calculations. This is more or less what is intended; water flowing to cells in a local depression gathers until the local depression is filled and then continues its way to lower situated areas. This means that water arriving at other cells in the local depression is able to flow to Q, before flowing to lower grounds. The calculations would now give a better overview of the potential watershed of this flooded area, and therefore a better overview of the potential watershed of the original set Q'. On the other hand, for example on a mountainous area, the extension of the set Q can include cells that are not significant for the current area of interest. Let us take, for instance, the mountain sketched in Figure 8.10b. The cells that are positioned in the grid on the place of the isolines have an elevation represented by the height of that isoline. If we calculate the watersheds for the set $Q = \{c\}$, where c is positioned on the green isoline, the new algorithm would enlarge the set Q by all the cells on that green isoline. At that moment, we are not calculating the watershed for the area of cell c anymore, but we are calculating the watersheds for the complete mountain path of the height of cell c; This will return the potential watershed containing the cells of the complete mountain top above within the green isoline, which is absolutely not the result we intended to find.



(a) Sketch of the situation of single-cell behaviour

(b) Sketch of a set of isolines representing the top view of a mountain

Figure 8.10: Examples to clarify one of the drawbacks of the watershed algorithms used

8.2.2 Computation times

A second drawback of the (implementation of the) algorithms, is the fact that calculations for a complete area (even if the area contains not that many cells) costs a lot of time. The theoretical running times of the algorithms are $O(n \log n)$ and $O(n^2 \log n)$. Unfortunately, the input terrains are very large — hundreds of millions cells — and the algorithms contain a lot of set calculations on this input. Taking the complement of a set, computing the difference of two sets; with really large sets, this takes a lot of time due to the fact that these sets do not fit in memory any more. Other calculations, such as the expansion of a cell, are also delayed when the information needed by the algorithm is too much to fit in memory. At that moment, the I/O-inefficiency of an implementation becomes important. There is a large similarity between the computation of a potential watershed (also for the Q-avoiding potential watershed) and a regular Dijkstra single-source shortest path query. The only difference is the way the 'distance' is calculated; for watersheds we investigate the possibility of a flow path rather than computing the length of the path. This difference is captured in the EXPAND functions of the watershed algorithms.

Henzinger et al. [HKRS97] propose a (theoretical) linear time algorithm for computing a Dijkstra query, by changing the order in which the nodes are extracted from the queue — with the possibility that a node is extracted multiple times. The structure of this algorithm is very similar to the structure of a standard I/O-efficient algorithm — dividing the area in smaller partitions, compute answers for the smaller partitions and combine the results — we suspect that this structure can be used to obtain an I/O-efficient implementation of the Dijkstra algorithm. Since the only difference between the Dijkstra query and a watershed query is the way the 'distance' is determined, and the queueing part is the same, we also believe that this approach can be used to speed-up the watershed algorithms. One interesting question for further research would be, whether this theoretical linear-time algorithm can indeed be translated to an I/O-efficient technique for Dijkstra queries.

8.2.3 Flow on levelled terrain

Another problem could be the fact that water can flow to neighbours with an equal height. In most GIS computations, grids are pre-processed and areas with neighbouring cells with an equal elevation are perturbed such that equal heights for two neighbours do not occur any more. It is very difficult to say anything about the influence of this choice, because the algorithm calculates the watersheds based on ranges and it is not always clear when a decision is made for equal neighbour flow. Since it is possible to select any elevation value within the allowed range, independently of the elevations selected for the neighbouring cells, we think the dam-building effect occurring in the min-max method described in Section 7.1 can also occur in elevation-bounded models that are obtained by different interpolation methods, but on a more local level. The basic ingredient for this dam-building effect is that the ranges of the neighbouring vertices should overlap. It could be possible that allowing flow to neighbours with an equal height enlarges the possibilities to dam-building. However, further research should be conducted to confirm this claim.

8.2.4 Uncertainty captured

Although there are still some questions about the watershed algorithms for the elevation-bounded model, we should not forget that, when selecting an appropriate area Q (for instance, a local minimum), we can compute the region for which water will certainly end up in Q and the region for which water may end up in Q. These watersheds take the elevation ranges into account and and give more information on the possibilities of the terrain compared to calculations on models with fixed elevations. The areas of the potential watershed that are not included in the persistent watershed are then areas for which the uncertainty in the elevations of the cells can give no definite conclusion on which flow path for ending up in the ocean is followed on the real terrain. The persistent watershed then gives a good impression of the minimal size of the watershed and the possible areas from which water can end up in Q as well.

This adds value to existing methods, which base their calculations on fixed elevations and do not take into account that the terrain may have a different shape in reality. Furthermore, by fitting a plane or polynomial surface to the area to smooth the terrain in a sense, we deal with measurement errors and are able to see the results of both situations.

8.3 Further research

Based on the certainty results and the discussion of the single-cell behaviour of the algorithms, we conclude that a simple calculation of the watersheds of a (set of) cells is not sufficient to base any conclusions on, since the results heavily depend on the behaviour of direct neighbouring cells. Since the suggested approach for extending the set of cells has the drawback that original intentions of a calculations get lost, a new way of determining how to extend a set Q should be formulated. This can

be done by estimating a sort of measure for the probability that a cell belongs to the area of interest or not, and based on these estimates, for example, extending the set Q. It would be very useful to put some time in investigating the possibilities for this approach, therefore, we suggest this for further research.

As explained in Section 4.2.5, it could also be useful to investigate the definitions of a persistent watershed. We are not sure whether it makes sense that the persistent watershed of a set Q can be disconnected, taking the idea behind flooding into account. The new definition however, should be weaker than the original definition of the core watershed, as already explained by Driemel et al. [DHLS11]. Based on the new definition, experiments should be conducted on real terrains to find out whether the definition has the desired results.

The algorithm for calculating the Q-avoiding potential watershed of a set Q for the MFD model takes $O(n^2 \log n)$ time, as proven in Section 4.2.4. This also affects the running time of the persistent watershed of Q in the MFD model, since this algorithm invokes the Q-avoiding potential watershed algorithm. We are interested whether there exists a faster way to calculate the Q-avoiding potential watershed and the persistent watershed of Q. One could think of a complete new approach, by replacing the method for finding the peak avoid watershed by a simpler one if possible, or by enhancing the existing algorithms. Future research should show the possibilities in this matter.

Other questions that arose during this project, which are interesting for further research are:

1. Is there a way to pre-process LiDAR data to a grid with useful uncertainty ranges?
2. Is it a problem that water can flow to neighbours with equal height?
3. How can we adapt the watershed algorithms such that the single-cell behaviour has less impact on the results?
4. Is it possible to use the techniques in the in theory linear-time algorithm as proposed by Henzinger et al. [HKRS97] for an I/O-efficiency implementation in practice?

Nomenclature

$p \overset{\rightarrow}{}_{R} q$	p flows to q in R
$\mathcal{T}$	Representation of an imprecise terrain
π	A flow path
$\Pi(S)$	The set of all flow paths in any realization in S, $S \subseteq \mathcal{R}_{\mathcal{T}}$
$\pi[p,q]$	The subpath of π from p to q
$\mathcal{R}_{\mathcal{T}}$	The set of all realizations of an imprecise terrain $\mathcal{T}$
$\mathcal{W}_R(Q)$	Watershed (SFD)
$\mathcal{W}_{\cap\!\cdot}(Q)$	Persistent watershed (SFD)
$\mathcal{W}_{\cap}(Q)$	Core watershed (SFD)
$\mathcal{W}_{\cup}(Q)$	Potential watershed (SFD)
$\mathcal{W}_{\cup}^{\setminus Q}(S)$	Q-avoiding potential watershed (SFD)
$\underset{MFD}{\mathcal{W}_{\cup}^{\setminus Q}}$	Q-avoiding potential watershed (MFD)
$\underset{MFD}{\mathcal{W}_R}$	Watershed (MFD)
$\underset{MFD}{\mathcal{W}_{\cap\!\cdot}}$	Persistent watershed (MFD)
$\underset{MFD}{\mathcal{W}_{\cup}}$	Potential watershed (MFD)
$N(P)$	The neighbourhood of a point set P
3-CNF	Conjunctive Normal Form formula with clauses of three variables
3-SAT	3-Satisfiability Problem
DEM	Digital Elevation Model
GIS	Geographic Information Science

MFD Multiple Flow Direction

SFD Single Flow Direction

Bibliography

[CZL12] Qin Cheng-Zhi and Zhan Lijun. Parallelizing flow-accumulation calculations on graphics processing units — From iterative DEM preprocessing algorithm to recursive multiple-flow-direction algorithm. *Computers and Geosciences*, 43(0):7–16, 2012.

[Dav08] Laurie Davies. Linear Regression and the Analysis of Variance. Lecture notes, Eindhoven University of Technology (TU/e), 2008.

[DHLS11] Anne Driemel, Herman Haverkort, Maarten Löffler, and Rodrigo Silveira. Flow Computations on Imprecise Terrains. *CoRR*, abs/1111.1651, 2011. v2, last revised 26 Sep 2012.

[dPCRM08] Adriano da Paz, Walter Collischonn, Alfonso Risso, and Carlos Mendes. Errors in river lengths derived from raster digital elevation models. *Computers and Geosciences*, 34(11):1584 – 1596, 2008.

[FT06] Peter Fisher and Nicholas Tate. Causes and consequences of error in digital elevation models. *Progress in Physical Geography*, 30(4):467–489, 2006.

[GKLS12] Chris Gray, Frank Kammer, Maarten Löffler, and Rodrigo Silveira. Removing Local Extrema from Imprecise Terrains. *Computational Geometry*, 45(7):334 – 349, 2012.

[GP08] Stephan Gruber and Scott Peckham. Land-Surface Parameters and Objects in Hydrology. In *Geomorphometry*, volume 33 of *Developments in Soil Science*, pages 171–194. Elsevier, 2008.

[HKRS97] Monika Henzinger, Philip Klein, Satish Rao, and Sairam Subramanian. Faster shortest-path algorithms for planar graphs. *J. Computer and System Sciences*, 55(1):3–23, 1997.

[HTar] Herman Haverkort and Laura Toma. Terrain Modeling for the Geosciences. In Teofilo Gonzalez, editor, *Computing handbook set*, volume I: Computer Science. CRC Press, To appear.

[Jan08] Jeffrey Janssen. Drainage computations on Digital Elevation Models. Master's thesis, Eindhoven University of Technology (TU/e), 2008.

[Lin06] John Lindsay. Sensitivity of channel mapping techniques to uncertainty in digital elevation data. *International Journal of Geographical Information Science*, 20(6):669–692, 2006.

[Mon09] Douglas Montgomery. *Design and Analysis of Experiments*. Wiley, 7th edition, 2009.

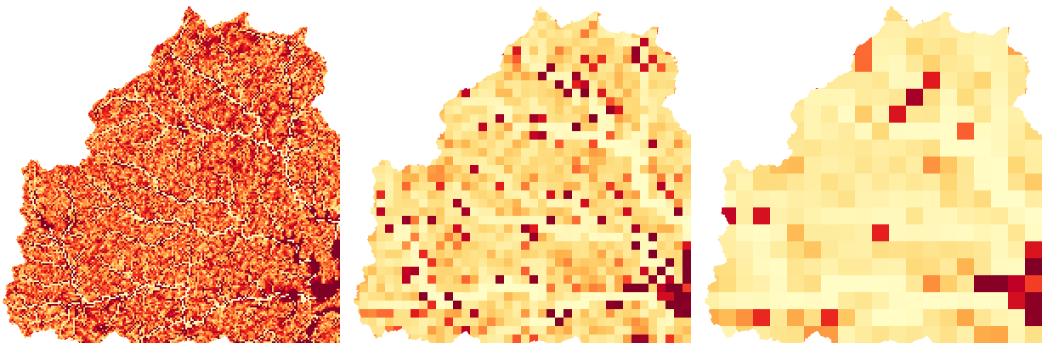
[Wis07] Steven Wise. Effect of differing DEM creation methods on the results from a hydrological model. *Computers and Geosciences*, 33(10):1351–1365, 2007.

[ZL02] Qiming Zhou and Xuejun Liu. Error assessment of grid-based flow routing algorithms used in hydrological models. *International Journal of Geographical Information Science*, 16(8):819–842, 2002.

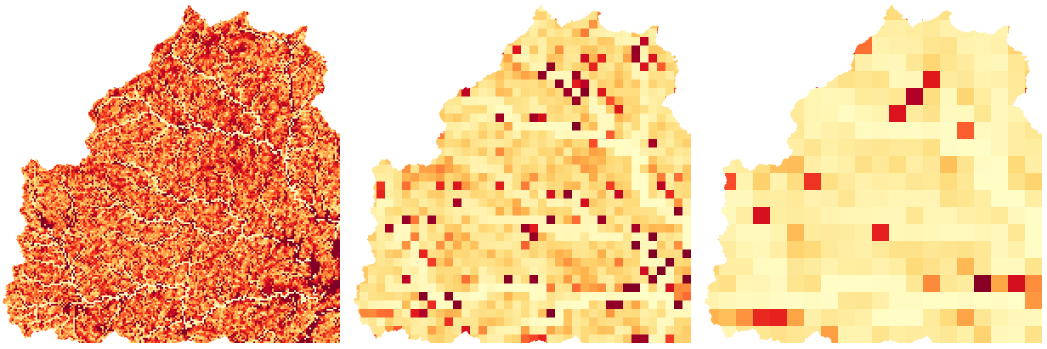
APPENDIX A

Extra results

In this appendix, more results supporting the claims in Chapter 8 are presented.

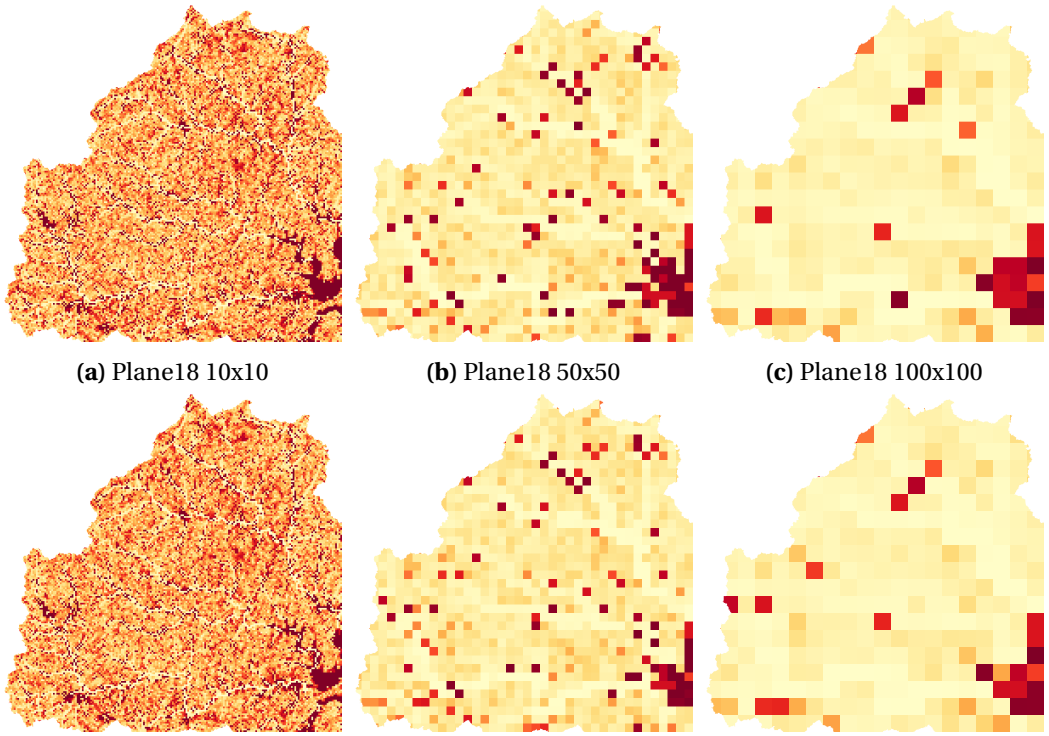


(a) Plane30 10x10 **(b)** Plane30 50x50 **(c)** Plane30 100x100



(d) PlaneSurf30 10x10 **(e)** PlaneSurf30 50x50 **(f)** PlaneSurf30 100x100

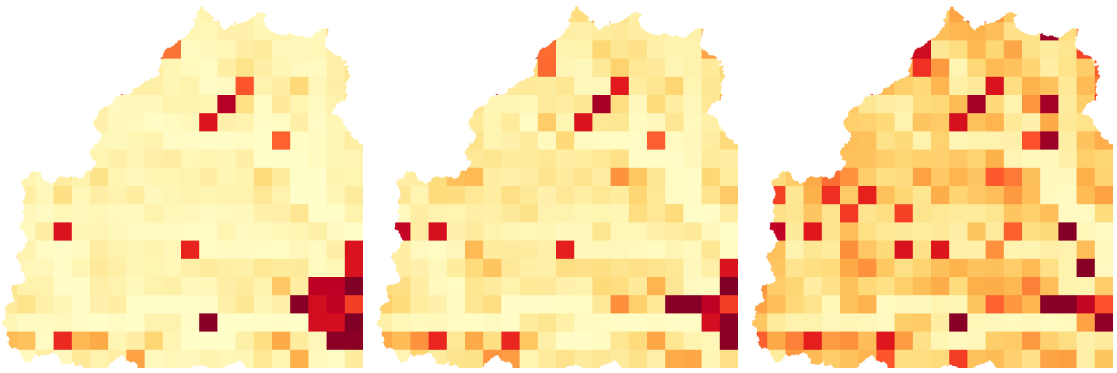
Figure A.1: Results of the different interpolation methods on the Neuse data set for the 30 neighbours setting and the SFD method.



(a) Plane18 10x10 **(b)** Plane18 50x50 **(c)** Plane18 100x100

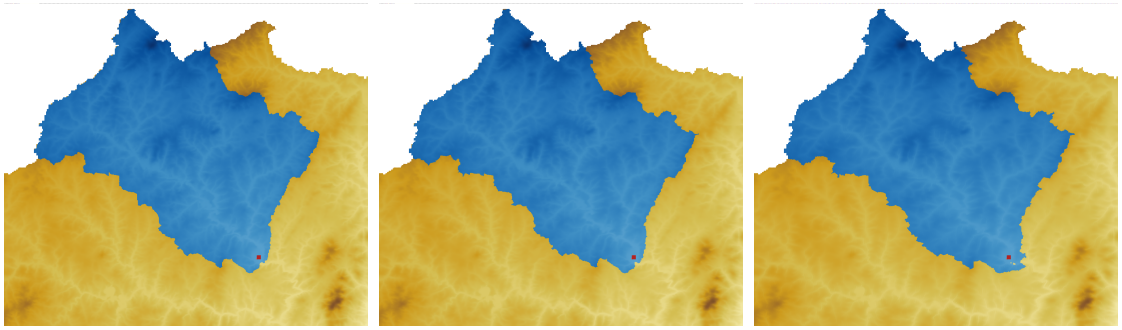
(d) PlaneSurf18 10x10 **(e)** PlaneSurf18 50x50 **(f)** PlaneSurf18 100x100

Figure A.2: Results of the different interpolation methods on the Neuse data set for the 18 neighbours setting and the SFD method.



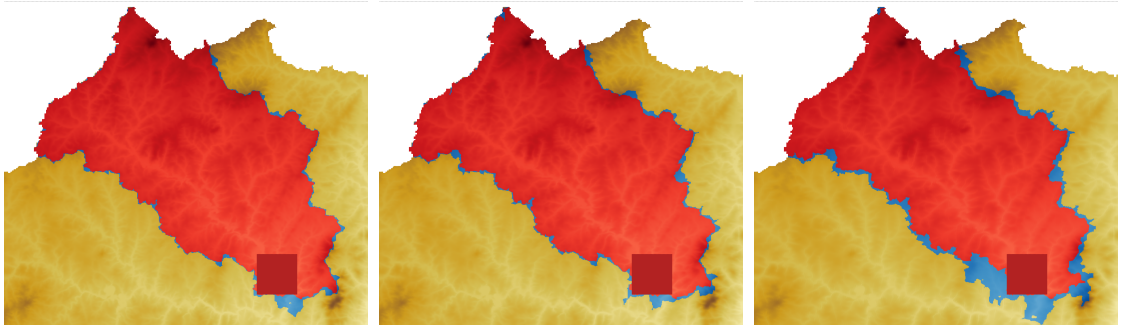
(a) Plane18 100x100 **(b)** Plane30 100x100 **(c)** Plane72 100x100

Figure A.3: Interpolating based on smaller neighbourhoods results in lower uncertainty



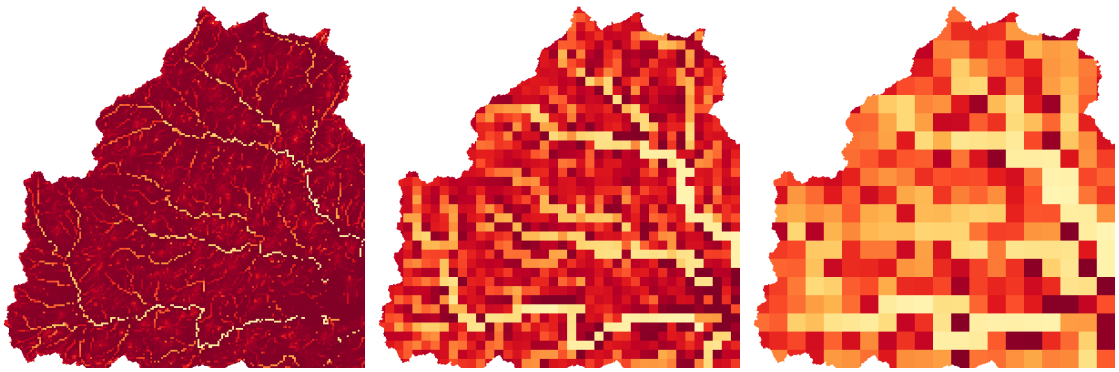
(a) Plane18 10x10 **(b)** Plane30 10x10 **(c)** Plane72 10x10

Figure A.4: Comparison of the results for different neighbourhood settings for a block of size 10x10



(a) Plane18 100x100 **(b)** Plane30 100x100 **(c)** Plane72 100x100

Figure A.5: Comparison of the results for different neighbourhood settings for a block of size 100x100



(a) Plane72 10x10 **(b)** Plane72 50x50 **(c)** Plane72 100x100

Figure A.6: Results of the Plane interpolation method on the Neuse data set for the 72 neighbours setting and the MFD method.

This Page Intentionally Left Blank