Tracking the Rise and Fall of Riparian Vegetation: A Temporal Network Approach

Msc Thesis Report

Author: Timme van Bergeijk

Student number: 6948871

E-mail: t.m.p.vanbergeijk@students.uu.nl

1st supervisor: Dr. E.A. Addink

2nd supervisor: Dr. M.W. Straatsma

Department of Physical Geography

Faculty of Geosciences

Utrecht University

Date: 08-07-2025

Abstract

Fluvial landscapes are the product of complex interactions between physical and biological processes, including hydrological fluctuations, sediments transport dynamics, and vegetation. In recent years, vegetation dynamics have been increasingly recognized as a key factor shaping riparian landscapes. Traditionally, vegetation dynamics have been assessed by comparing land cover classes between timesteps, which captures broad trends but fails to track individual vegetation patches over time. This study addresses that gap by introducing a novel object-based temporal network framework that links classified land cover objects from 15 aerial image years (1946–2019) to reconstruct the lifecycles of large, high riparian vegetation (HV) patches on a stretch of the Allier River. Here we demonstrate the value of the novel approach, revealing spatial and temporal dynamics that are obscured in traditional class-based change analyses. The results show that most large HV patches follow a rise–and–fall trajectory, typically lasting around 10 years. Growth most often begins through emergence from bare soil or low vegetation, while further development occurs through merging and eating of nearby patches. Degradation usually occurs through direct disappearance to non-high-vegetation patches, like bare soil, low vegetation or water, while an intermediate step of patches falling apart to smaller HV patches is also observed. These patterns reveal key insights into patch-scale dynamics that are not accessible through traditional class-based methods. By shifting focus from aggregated land cover change to individual patch trajectories, this method reveals the temporal and spatial processes underlying vegetation development. The approach offers a tool for ecological monitoring and provides a basis for future predictive models that integrate vegetation dynamics with hydrological and geomorphological change.

Table of Contents

List of Figures

List of Tables

1. Introduction

1.1 Background riparian system dynamics

Fluvial landscapes are the product of complex interactions between physical and biological processes, including hydrological fluctuations, sediments transport dynamics, and vegetation dynamics (Bertagni et al., 2018; D'Alpaos et al., 2016). In recent years, vegetation dynamics have been increasingly recognized as a key factor in shaping riparian landscapes. Fluvial geomorphological processes, such as erosion, sedimentation, and meander migration, continuously reshape riverine environments. At the same time, riparian vegetation both responds to and influences these physical processes, creating what is known as a biogeomorphological system (Grabowski et al., 2022; Gurnell, 2014; Solari et al., 2016). In these systems, vegetation not only adapts to the conditions provided by river morphodynamics but can also actively modify its environment. Tree species, for example, are known to act as eco-engineers by enhancing conditions favorable for their own growth (Corenblit et al., 2015). The importance of this two-way interaction between fluvial processes and vegetation is increasingly recognized as an area of interest in understanding the dynamics of fluvial systems (Bertoldi et al., 2011).

Within floodplains, the dynamic nature of riparian vegetation is primarily governed by two key processes: succession and disturbances. Succession begins with the establishment of pioneer species on bare sediment surfaces, followed by progressive growth, aging, and the gradual development of more complex vegetation structures. This results in increased biomass, vegetation height, and density over time. During the pioneering stage of the succession, settlement of riparian vegetation is only governed by environmental variables. On the other hand, vegetation development in later stages of the succession is also governed by the spatial distribution of vegetation through facilitation and competition (Camporeale et al., 2013; Geerling et al., 2006). The process of vegetation succession is frequently interrupted by natural disturbances such as floods and droughts, which lead to vegetation mortality, patch fragmentation, or even complete removal of vegetation patches, a process known as rejuvenation (Bätz et al., 2016; Kui et al., 2014). A recent study by Mohan & Joseph (2024) emphasizes that riparian vegetation dynamics are shaped not only by large episodic events, but also by continuous background degradation, driven by small-scale disturbances and competitive interactions. As a result, riparian vegetation may advance toward a climax state, characterized by large, tall, and well-established patches, but it can also regress to earlier successional stages in response to disturbance.

These ongoing cycles of establishment, growth, and degradation result in a spatially heterogeneous floodplain consisting of vegetation at varying successional stages, from bare or early-colonizing surfaces to mature, high vegetation (Chuman et al., 2006). Although the spatial distribution of land cover classes at the landscape scale may appear relatively stable over time, this masks high levels of turnover and structural change at the patch level (Garófano-Gómez et al., 2017; Geerling et al., 2006). The rise of riparian forests is often facilitated by merging of adjacent vegetation patches, where new growth is more likely to occur near existing high vegetation than on isolated bare soil (Kleinhans et al., 2019). However, the precise patterns that describe the rise and fall of climax vegetation are yet to be fully understood. Despite increasing understanding of these processes, a significant knowledge gap remains regarding the spatial and temporal evolution of individual patches, particularly those that growth into or degrade from large, tall vegetation areas.

The relevance of these dynamics extends beyond ecology and geomorphology. Riparian vegetation influences critical ecosystem services such as flood attenuation, sediment trapping, habitat provision, and bank stabilization (Hubble et al., 2010; Riis et al., 2020; Rutherford et al., 2007). Hence, understanding how high vegetation patches rise and fall is vital for river management, restoration planning, and biodiversity conservation, especially under increasing anthropogenic and climatic pressures.

1.2 State of the art / Knowledge Gap

In order to monitor temporal evolution of vegetation dynamics, remotely sensed data is often used. Especially series of aerial photographs, with their high spatial resolution, is useful for studies trying to understand small-scale dynamics (Bertoldi et al., 2011). Current approaches to monitor riparian vegetation development primarily rely on static land cover maps derived from satellite or aerial imagery at individual time steps. For these individual timesteps, vegetation dynamics are often monitored by analyzing the proportion of the area of a certain land use class per year or by transition matrices between two consecutive time steps (Bertoldi et al., 2011; Douda et al., 2009). While these methods can quantify overall changes in land cover, they fall short in understanding the spatial and temporal complexity of riparian vegetation dynamics.

In this study, the focus lies specifically on large, high vegetation (HV) patches, defined as tall, late-successional vegetation patches exceeding 150 m^2 in area, dominated by mature trees and bushes and considered as postpioneer species (Kleinhans et al., 2019). Without a framework to track such individual patches through time, the ability to infer causal relationships between environmental drivers and vegetation outcomes is limited. As a result, ecologically meaningful processes, like the growth or degradation trajectory of HV patches, remain untraced.

However, current approaches are limited in their ability to track such dynamics. Most methods rely on class-based transitions between individual time steps and do not consider the continuity of vegetation patches through time. As a result, key processes such as emergence, expansion, merging, fragmentation, and disappearance remain unaccounted for. This makes it difficult to assess long-term trajectories of HV patches or the cumulative impacts of disturbances and biotic interactions. Without a framework that follows individual patches through time, ecologically meaningful patterns of growth and degradation remain untraced.

A recent study by Kleinhans et al. (2019) applied object based image analysis (OBIA) to investigate the short-term fate of small, isolated vegetation patches in a dynamic meandering river system (Allier River). Their approach used OBIA to classify water, bare soil, low vegetation, and high vegetation from aerial photographs and track small, isolated vegetation objects between three time steps, enabling them to identify key processes such as establishment, persistence, and removal. In Kleinhans et al. (2019) OBIA was chosen over pixel-based classification due to the limitations of spectral information of the aerial photographs, of which many were black-and-white. In such imagery, the spectral differences between land cover types are often insufficient for accurate classification at the pixel level. Moreover, pixel-based methods do not consider the spatial arrangement of pixels, typically resulting in fragmented, noisy maps that poorly reflect actual vegetation structures. OBIA overcomes these problems by first segmenting the pixels into meaningful objects based on both spectral and spatial characteristics. These objects are then classified using a range of features, including not only spectral properties but also geometric attributes (e.g., area, shape index, compactness), texture, and contextual relationships (e.g., contrast with neighboring objects) (Addink et al., 2012). These properties and attributes are specifically advantageous in riparian environments, with vegetation patches that are irregularly shaped, internally heterogeneous, and often interspersed with bare soil. OBIA captures this complexity by treating vegetation as spatial entities rather than disconnected pixels, allowing for more accurate mapping of vegetation structure and distribution. Finally, a key strength of OBIA is its ability to merge adjacent pixels into ecologically meaningful units that reflect observable features in the landscape, such as high vegetation and isolated vegetation patches.

While this work demonstrated the value of object-based methods for identifying early-successional vegetation dynamics, it focused primarily on small, short-lived patches, and relied on only three different time steps. As a result, it left open questions about the long-term development and degradation of larger, more stable vegetation structures that represent late-successional stages.

As mentioned, a key bottleneck in existing studies is the lack of methods that enable continuous tracking of individual patches across multiple time steps. Object-based classification provides a necessary foundation for this, as it represents each vegetation patch as a distinct spatial object. However, until now, no studies have explored how these objects can be linked through time using robust frameworks.

Therefore, to understand the dynamics of riparian systems, and more specifically the persistence, succession, or disappearance of HV patches in these systems, a methodological shift is required. This shift involves moving away from traditional class-based analyses that compare individual time steps in isolation, toward approaches that can track individual vegetation objects continuously through time.

1.3 Research Approach

To implement this methodological shift, this study introduces a novel method that combines Object-Based Image Analysis (OBIA) with temporal network modeling to reconstruct and analyze the spatiotemporal dynamics of HV patches. A temporal network is a type of graph-based data structure used to represent systems that change over time. In such networks, nodes represent entities at specific time points, and edges represent relationships or interactions that occur between them across time steps (Kivela et al., 2014). Temporal networks are therefore useful for analyzing how a system evolves through time.

In this study, the nodes in the network represent objects with classes, each corresponding to different stages in the ecological succession. The temporal network will be constructed using a classified time series of aerial photographs, based on the dataset used by Kleinhans et al. (2019) and expanded with four more recent aerial photographs. Additional years will be added by classifying more recent aerial imagery using OBIA, and the new classifications will be validated through an accuracy assessment.

By linking HV patches across multiple years, the network enables a spatio-temporal analysis of patch history, offering insights into the trajectories of individual patches rather than just aggregate change. Then, the progression toward HV patches, as well as their potential subsequent degradation (rise and fall) will be looked into in more detail by analyzing specific transition types (e.g. emergence, expansion, merging, fragmentation, shrinking), which together represent the key dynamics of vegetation patch development and decline.

1.4 Research Questions

This research focuses on the spatial and temporal dynamics of large, high riparian vegetation (HV) patches within a floodplain context. The central research question guiding this study is:

How do HV patches emerge, grow, and degrade over time in a dynamic fluvial landscape?

To answer this central question, this study addresses the following sub-questions:

- How long do HV patches typically persist?
- What are the dominant patterns of growth and degradation in HV patches, and how do these evolve throughout a patch's lifespan?
- How do landscape-scale river dynamics influence the development of HV patches?

1.5 Study area

The study area is a section of the Allier River in central France, situated between Moulins to the north and Châtel-de-Neuvre to the south (**Figure 1**). The rain-fed Allier River has its source in the Massif Central and debouches into the Loire River near Nevers. The mean annual discharge is 140 m^3/s, and the mean peak discharge is 500 m^3/s (van Dijk et al., 2014). The Allier River is often used for (eco-) morphodynamic research due to its unconfined lateral morphology, which allows for natural meandering (Geerling et al., 2006). Riparian vegetation strongly interacts with the meandering process in the Allier River, resulting in a valuable system to explore the role of riparian vegetation and river morphology on each other. The selected 4.3 kilometers long stretch of the Allier River used for this study comprises four to six active meander bends, providing areas with multiple stages in the succession of riparian vegetation. The combined river and floodplain area extends to an area of 286 ha and contains areas with bare soil (sandy gravel) and riparian vegetation (grasses, herbs, shrubs, and forest) (Garófano-Gómez et al., 2017). Given its dynamic morphology and the availability of high-resolution aerial imagery, the Allier River offers an ideal setting to study the rise and fall of riparian vegetation patches across multiple decades.

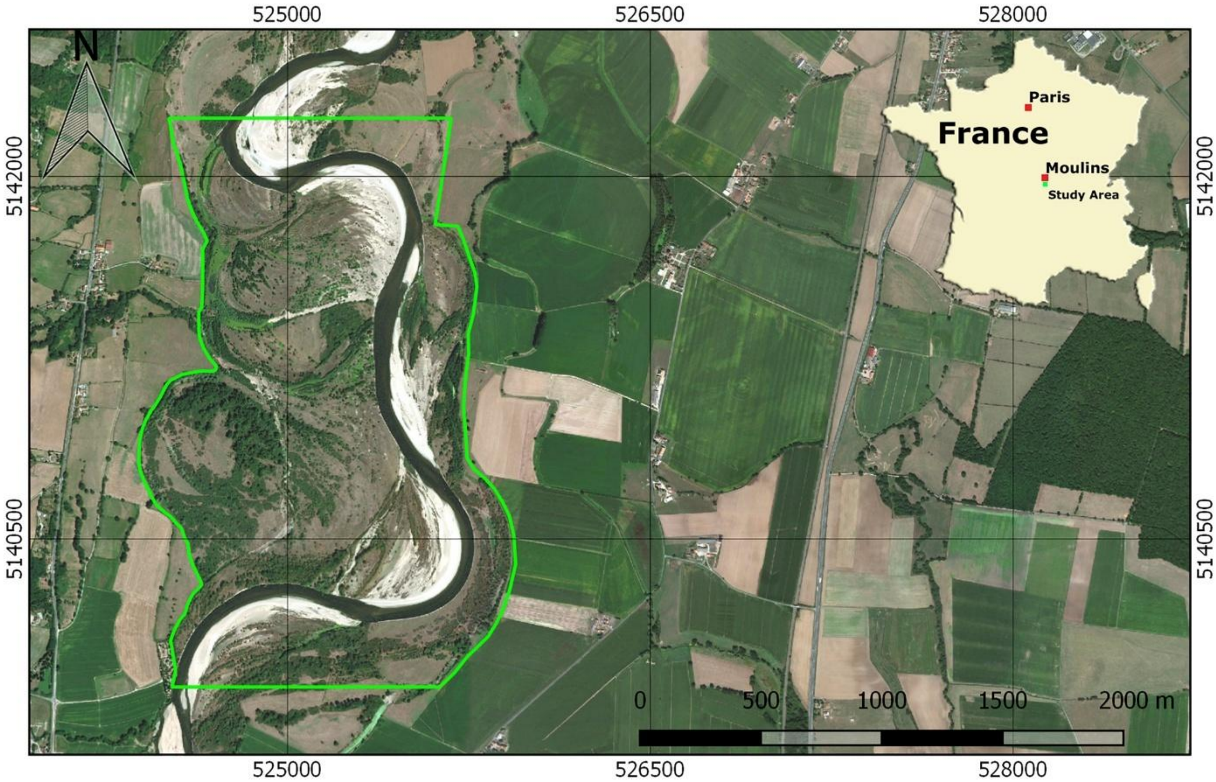


Figure 1. Study area in the Allier River (Kleinhans et al., 2019).

2. Data and Methods

To analyze the spatial and temporal development of large riparian vegetation patches, this study combines object-based image analysis (OBIA) with a novel temporal network framework. The input for the temporal network is the classified time series from Kleinhans et al. (2019) with the addition of four more recent aerial images. The new images are classified using the same OBIA approach as in Kleinhans et al. (2019). Unlike traditional land cover change methods that compare class areas between years, this network approach enables tracking of individual vegetation objects across time steps. This object-level continuity allows for reconstruction of vegetation patch lifecycles and detailed interpretation of growth and degradation dynamics. The overall methodological workflow consist of five steps and is shown in **Figure 2**. In the Appendix, a detailed flowchart is added with all methodological steps.

Figure 2. Overview of methodological steps.

2.1 Data preparation

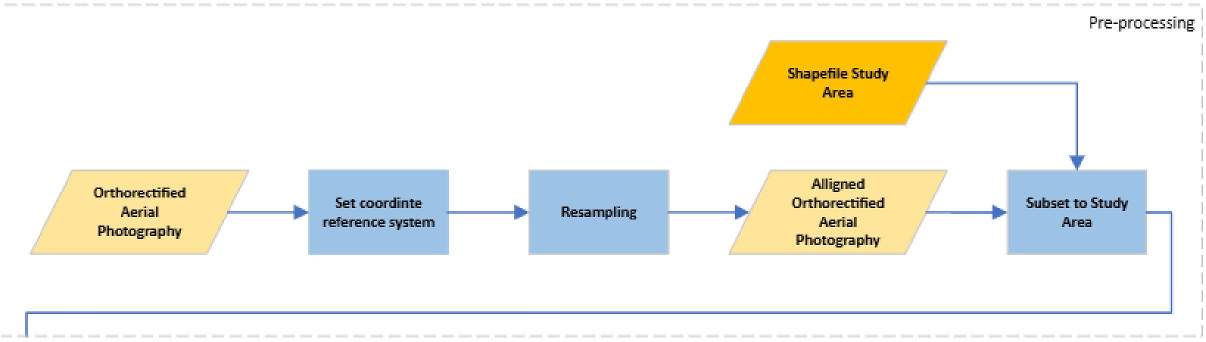


Figure 3. Flowchart with methodological steps for data preparation.

For this thesis a combination of both already classified maps by Kleinhans et al. (2019) spanning over 50 years (1946, 1954, 1960, 1968, 1975, 1980, 1985, 1992, 1998, 2000, 2002) and 4 new, yet to classify maps (2009, 2013, 2016, 2019) are used. These four new orthorectified images are obtained from the database from the Centre Régional Auvergne-Rhône-Alpes de l'Information Géographique (CRAIG) (2025). All aerial images were captured during the summer months, ensuring consistent vegetation phenology and lighting conditions across all years. This seasonal consistency enhances comparability and reduces the likelihood of misclassifications due to seasonal variation. The four new images were clipped to the study area and resampled to the coordinate system WGS 1984 UTM 31N and to a spatial resolution of 0.3 meters. As the 11 already classified maps by Kleinhans et al. (2019) also have variable resolutions, before adding them to the network analysis, they were resampled to 0.3 meter as well (Table 1).

Table 1. Aerial photographs used for analysis.

Image date	**Color**	**Resolution (m)**	**Resampling (m)**
13/07/1946	BW	0.92	0.3
09/05/1954	BW	0.92	0.3
03/06/1960	BW	0.92	0.3
30/06/1968	BW	0.264	0.3
28/07/1975	BW	0.188	0.3
02/08/1980	BW	0.323	0.3
09/07/1985	BW	0.34	0.3
31/07/1992	BW	0.32	0.3
20/06/1998	BW	0.275	0.3
19/07/2000	RGB	0.265	0.3
31/05/2002	RGB	0.76	0.3
??/08/2009	RGB	0.3	-
02/08/2013	RGB	0.25	0.3
26/08/2016	RGB	0.25	0.3
15/09/2019	RGB	0.2	0.3

2.2 Object Based Classification

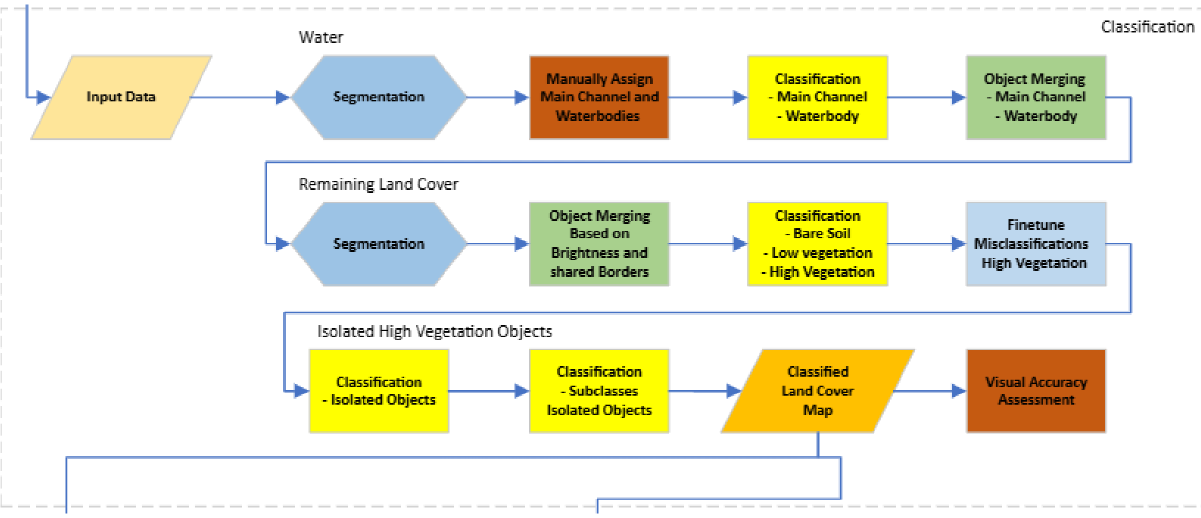


Figure 4. Flowchart with methodological steps for OBIA classification.

The four added aerial images are segmented and classified using OBIA in eCognition software (Trimble, 2025). The classification approach is based on the method developed by Kleinhans et al. (2019), who used OBIA to map riparian land cover classes, with the focus on small, isolated vegetation patches. Their workflow involved segmenting images into homogeneous objects, classifying these based on brightness and shape metrics, and distinguishing isolated vegetation patches from broader vegetation patterns. This workflow was copied with some refinements that focus on an optimal classification of large, high vegetation patches. In Table 2, the classified land cover classes are presented with a short definition of their properties.

The first step on OBIA is segmenting the aerial images. As the pixel sizes (after resampling) and spectral values for the aerial images of 2009, 2013, 2016, and 2019 were comparable, the segmentation parameters were set to the same values for each year (Table 3). After testing multiple combinations of segmentation parameters to optimally delineate vegetation from bare soil and water, the segmentation algorithm was applied by using scale 35 with shape and compactness of 0.1 and 0.5 respectively. This segmentation approach results in a mosaic of objects.

After segmentation, first, the water classes were labeled manually. The reason for labeling these classes manually is the fact that water and high vegetation both have low brightness values, resulting in spectral confusion. After classifying the water objects, the remaining area is segmented again, and objects were merged based on three conditions regarding their common border, texture difference, and spectral difference. Objects were merged if (1) at least 5% of their borders were shared, and if (2) the standard deviation of brightness within the objects was maximum 2 for 2009 and 4 for the other three years, and if (3) the average brightness of the objects differed less than 12 (Table 3). This merging step helps reduce over-segmentation, where a single land cover unit may be split into multiple small, artificial objects, and ensures that each object more accurately represents a real-world unit, for example one large vegetation patch. Hereafter, the classes bare soil, low vegetation, and high vegetation were classified fully automatically based on brightness values of those three classes. Thresholds were set visually for each year to optimize the classification of each object to the correct class (Table 3). Besides the basic thresholds using brightness values, extra rules are added to finetune clear misclassifications. These finetune rules use parameters such as relative borders, contrast, shape, and brightness values (Appendix B. Finetune Rules OBIA). Lastly, isolated vegetation objects were distinguished from large vegetation objects. Kleinhans et al. (2019) defined large vegetation patches as larger than 150 m^2. Therefore, isolated vegetation patches are defined as smaller than 150 m^2. When isolated vegetation objects were extracted from low vegetation, the maximum area was set to 50 m^2 to minimize false positives. Isolated vegetation patches are characterized by the fact that they are darker than their surroundings and by having a roughly round shape. Hence, objects are only classified as isolated when their surrounding objects are all brighter and when their shape index is smaller than two. To add more insights into the development of vegetation patches in riparian environments, the isolated objects are divided into small and large objects, with a threshold of 30 m^2. The results are four isolated object classes: small and large isolated objects surrounded by bare soil, and small and large isolated objects surrounded by low vegetation. The classified objects are exported in one shapefile for each year with relevant information in the corresponding attribute table.

Table 2. Land cover classes with their definitions.

Class	*Definition*
MainChannel	Main river channel, permanently flowing
Waterbody	Body of water not connected to the main channel, e.g. oxbow lake
BareSoil	Exposed soil or sand with no vegetation cover
LowVegetation	Vegetated areas dominated by low herbaceous plants or grasses
HighVegetation	Patch with tall vegetation such as shrubs, bushes, or trees.
IsolatedSmallLowVeg	Small isolated high vegetation patch surrounded by low vegetation
IsolatedSmallBareSoil	Small isolated high vegetation patch surrounded by bare soil
IsolatedSmall	Small isolated high vegetation patch surrounded by both bare soil and low vegetation
IsolatedLowVeg	Isolated high vegetation patch surrounded by low vegetation
IsolatedBareSoil	Isolated high vegetation patch surrounded by bare soil
IsolatedObject	Isolated high vegetation patch surrounded by both bare soil and low vegetation

Table 3. Parameters object based classification in eCognition.

Category		*2009*	*2013*	*2016*	*2019*
Image Segmentation	Pixel size (m)	0.3	0.3	0.3	0.3
	Scale	35	35	35	35
	Shape, compactness	0.1, 0.5	0.1, 0.5	0.1, 0.5	0.1, 0.5
Object Fusion	*minimum_common_border*	0.05	0.05	0.05	0.05
	maximum_texture_difference	2	4	4	4
	Maximum_spectral_difference	12	12	12	12
Brightness Thresholds	*Bare soil – low vegetation*	175	163	156	154
	Low vegetation – high vegetation	112	105	115	110
Isolated Object Thresholds	*From high vegetation (area in m^2)*	<150	<150	<150	<150

From low vegetation (area in m^2)	<50	<50	<50	<50
Relative border to brighter object	1	1	1	1
Shape Index	<2	<2	<2	<2
Threshold small/large (area in m^2)	<30	<30	<30	<30

2.3 Accuracy Assessment OBIA

To evaluate the performance of the OBIA classification, an accuracy assessment was conducted. Due to the lack of available ground truth data for the dates used in the classification, an accuracy assessment was performed based on visual interpretation instead of a traditional validation using reference data. The implemented method is based on a comparison between the classified map and high-resolution aerial imagery. Only the four newly classified maps are assessed on accuracy. The classification from the years 1946 to 2002 are considered accurate in this thesis, as they are used for the peer reviewed paper from Kleinhans et al. (2019).

From the classified map, 100 random objects were selected and subsequently assessed through visual interpretation. This process was conducted in ArcGIS Pro 2.8.2 using a custom Python script within the ArcGIS Python window. The script initially stratifies the classification shapefile in 100 intervals based on object size. From every interval, one random object is selected, resulting in 100 random objects of assorted sizes. With this it is made sure that larger objects are also represented. These 100 randomly selected objects were exported as separate shapefiles for further evaluation.

After exporting the random selection, a new text field "observed" was added to the attribute table. The original classification field was hidden to avoid bias during the visual inspection. Each of the 100 objects was then visually assessed using the corresponding aerial image of that year and manually labeled by the observed class in the new text field.

Following this visual assessment, summary statistics were calculated in ArcGIS Pro to compare the original classified class and the observed (visually interpreted) class. Since each classified unit is an object with a defined area, two types of accuracy assessments were performed. Accuracy was calculated using the number of objects falling into each class, and by the total area of objects in each class combination. This second approach gives more weight to larger objects and provides insight into the classification performance in terms of spatial coverage. This dual approach allows for a more nuanced assessment of classification performance by considering both the number and the spatial extent of classified objects.

The resulting summaries were used to construct confusion matrices in Excel, from which standard accuracy metrics were derived. For the object count-based analysis, overall accuracy was calculated as the proportion of visually interpreted objects that were correctly classified in the OBIA output. Producer's accuracy, calculated per class, represents the proportion of reference (visually assigned) objects that were correctly identified by the classification. Conversely, user's accuracy indicates the proportion of classified objects that correctly correspond to the visual reference class. In the area-based analysis, overall accuracy was computed as the proportion of the total area that was correctly classified. Producer's accuracy, in this context, refers to the proportion of the reference area correctly classified, while user's accuracy reflects the proportion of the classified area that matches the reference class (Congalton, 1991).

2.4 Temporal Network

Where earlier studies on monitoring vegetation development focused on individual time steps, in this study a novel method is introduced that links vegetation patches across multiple years. This method combines OBIA with a temporal network. By linking vegetation patches across multiple years, the network enables a continuous analysis of patch lifecycles, offering insights into the trajectories of individual patches rather than just aggregate change. These reconstructed lifecycles capture both the rise (emergence and growth) and fall (degradation and disappearance) of individual vegetation patches and can be linked to environmental variables to better understand the conditions that drive the most frequently occurring pathways toward or from HV patches.

2.4.1 Set-Up Temporal Network

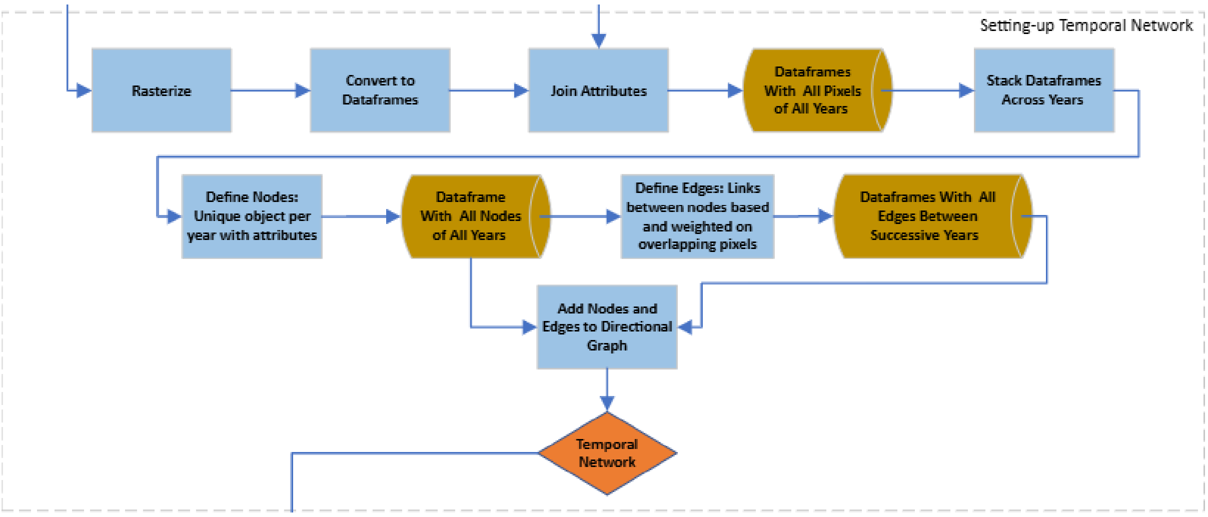


Figure 5. Flowchart with methodological steps for setting-up the temporal network.

A temporal network is a graph-based structure in which nodes represent entities at specific time points, and edges represent connections between those entities across time steps. In this study, each node corresponds to a unique vegetation object (from OBIA classification) in a specific year, and stores attributes such as class, area, and spatial location. Edges are formed between nodes in consecutive years when their corresponding objects share overlapping pixels, representing temporal continuity or transformation. This network structure, as introduced in section 1.3, enables detailed tracking of vegetation patch dynamics over time. Not only does a temporal network enable detailed tracking of HV patches over time, it also provides a framework for analyzing the underlying dynamics of HV growth and degradation.

The full script to create the temporal network is added in Appendix E. The first step in creating the temporal network is preparing the input data. Since the network operates on raster data, the classification shapefiles, each containing land cover objects, are first rasterized.

Then, the rasterized classifications are added to the script and converted to dataframes, where each row represents a pixel with corresponding object ID. To enrich the pixel-level data, these dataframes are joined with their corresponding attributes from the original shapefile classifications to get all the valuable information of the objects, such as class name and object area. While OBIA allows for the extraction of a wide range of object-level attributes, such as shape metrics or relative spatial context, these were not applicable to the scope of this study. All individual dataframes, each representing a year of classification, are then stacked and added to a single dataset.

At this stage, there is a dataset where every row represents a pixel of a certain year, with corresponding coordinates, object ID's, class names, and areas. This dataset is the source from which the nodes are defined. Each node corresponds to a unique object in a specific year and stores attributes that are relevant for understanding vegetation dynamics, such as location, size, and class.

To capture how the objects evolve over time, edges are created between nodes in consecutive years. An edge is created between a node in year *t* and a node in year *t+1* when they have overlapping pixels, based on the object's raster representation. This spatial overlap implies a temporal continuation or transformation of the object.

By connecting the nodes with edges, a directional graph is constructed. This directional graph models the temporal evolution of vegetation patches over time. Each node in the graph represents a unique object in a specific year and is identified by a tuple (year, Object_ID). Edges in the graph represent transitions between patches across consecutive years and are weighted by the number of pixels that link two patches over time. This structure enables the reconstruction of HV lifecycles and supports detailed analysis of the rise and fall of these patches.

2.4.2 Defining Transition Types

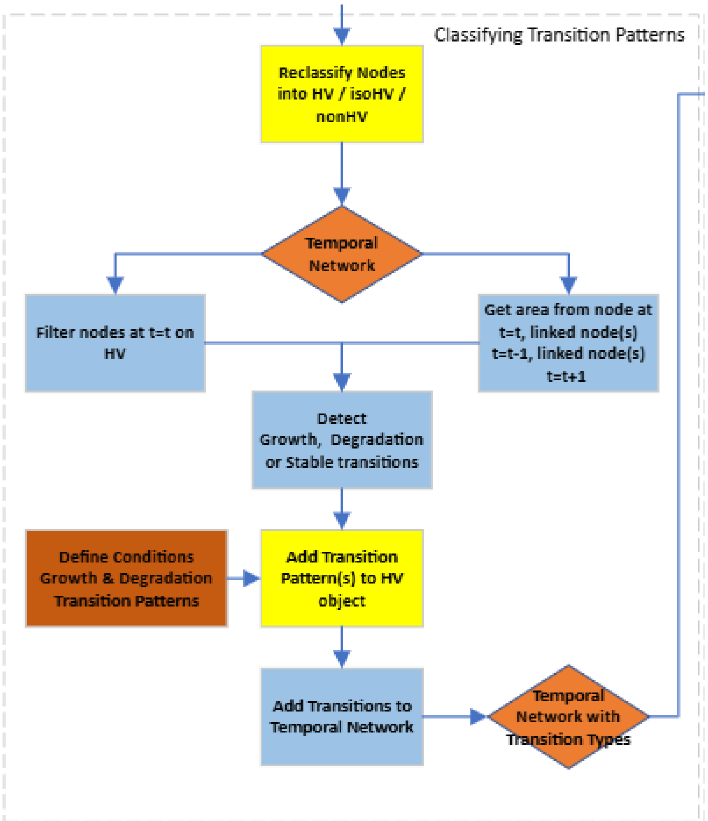


Figure 6. Flowchart with methodological steps for defining the transition types.

The transitions to or from HV patches can be described by the growth and degradation types, introduced by van Hemel (2019) and illustrated in Figure 7 & Figure 8.

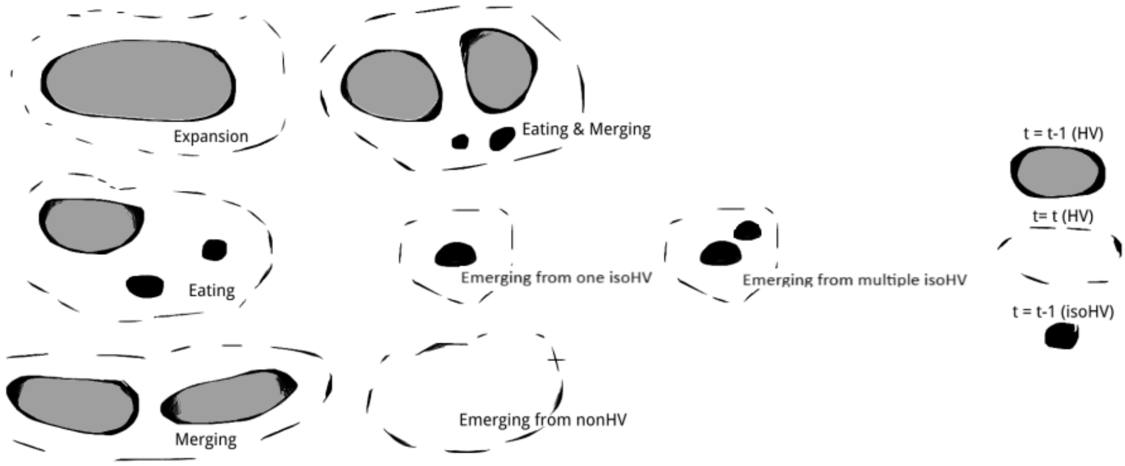


Figure 7. Sketch of growth transitions, adapted from van Hemel (2019). *HV = high vegetation, isoHV = isolated high vegetation*

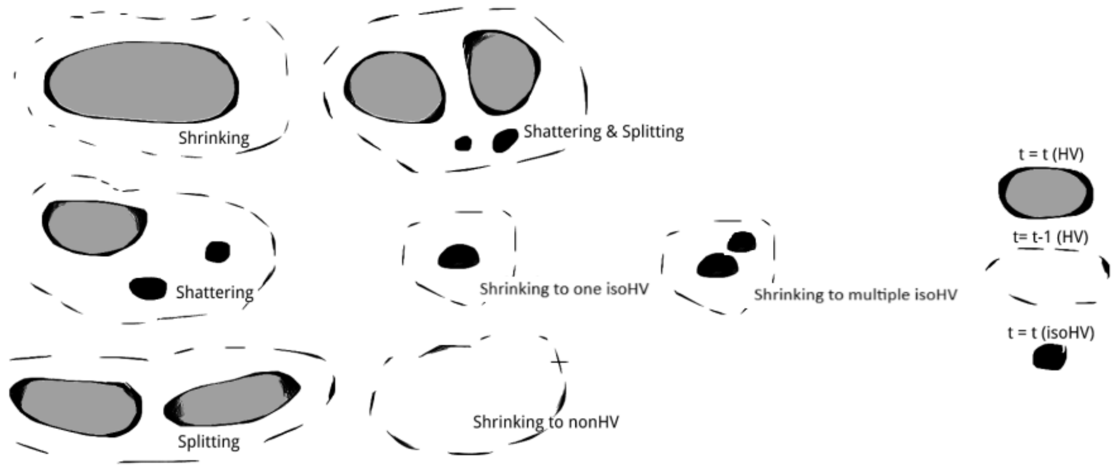


Figure 8. Sketch of degradation transitions, adapted from van Hemel (2019). *HV = high vegetation, isoHV = isolated high vegetation*

The transition types reflect real-world vegetation dynamics in riparian environments. Expansion refers to an HV patch increasing in size compared to the previous timestep by gradual outward growth into neighboring bare or low vegetation areas. Eating describes a process where small, isolated patches of high vegetation become part of a growing HV patch. Merging occurs when multiple HV patches, previously separate, connect and form a single larger unit. When both isolated and large HV patches are incorporated simultaneously, this is referred to as Eating & Merging. Emerging transitions occur when a new HV patch appears in a location where no HV existed in the previous timestep. If the new patch originates from small isolated HV fragments, it is classified as emerging from isolated HV; if it appears on locations that were bare soil, low vegetation, or water in the previous time step, it is emerging from non-high vegetation, typically reflecting early successional establishment.

On the degradation side, Shrinking captures the gradual reduction of HV area. Shattering occurs when parts of the HV patch break into smaller isolated HV patches. Splitting is similar but results in multiple HV patches that still exceed the threshold to be considered large. Shattering & Splitting represents a combination of both processes, where part of a HV patch fragments into small isolated HV patches and parts remain as HV patches, but reduced in size. When a HV patch degrades into only isolated vegetation, this is labeled Shrinking to isolated HV, and if it degrades entirely into low vegetation, bare soil, or water, it is classified as Shrinking to nonHV. These degradation types reflect disturbance-driven retreat or succession reversal due to fluvial activity, competition, or environmental stress.

A theoretically possible outcome is that the area of a HV object in year t is the same as in year t-1 or t+1. In that case the transition type is labeled as stable.

The temporal network is a perfect base to classify the transition types. Since the analysis is conducted from the perspective of HV patches, transitions are interpreted relative to the temporal development of these individual vegetation objects. Growth is examined retrospectively, looking from time nodes in t−1 that are connected with edges to node t, in order to determine how a HV patch may have formed or expanded from earlier HV objects. In contrast, degradation is assessed prospectively, from time t to t+1, to evaluate how a given patch may shrink, fragment, or disappear in the subsequent time step. In other words, growth focuses on the rise of a patch by looking one step backward in time, while degradation considers its potential fall by looking one step forward. The python script that is used to classify the transition types is added In Appendix F.

First, all transitions involving HV objects are classified to “growth”, “degradation”, or “stable”. For growth analysis, the algorithm traced backward in time by comparing each patch to its predecessors (i.e., from t-1 to t). A transition was classified as "growth" if the current patch represented High Vegetation and either did not originate from a High Vegetation patch or had increased in area compared to its High Vegetation predecessor(s). A similar logic was applied when tracing forward in time for degradation detection (i.e., from t to t+1), comparing each patch to its successors. If a High Vegetation patch transitioned to a non-High Vegetation state or decreased in area while staying High Vegetation, this indicates degradation. High Vegetation objects that remain the same area are classified as “stable”.

The next step is reclassifying the object classes. As can be seen in **Figure 7** & **Figure 8**, only three classes are used in describing the transition types. These three classes are HV (Large, High Vegetation objects), isoHV (Isolated High Vegetation Objects, which are all objects classified as isolated), and nonHV (non-High Vegetation, which are the objects classified as bare soil, low vegetation, main channel, and waterbody). With these classes and the fact that a transition is labeled as growth or degradation (or stable), the transition types from **Figure 7** & **Figure 8** can be defined.

The transition types are shown in **Table 4** for growth transitions & **Table 5** for degradation transitions. Here, the conditions are mentioned to determine the specific transition types, based on the classes and counts of predecessor nodes for growth and successor nodes for degradation. Using these conditions, all nodes in the temporal network representing HV patches are assigned a growth or degradation transition type. When a HV patch is formed through growth between *t–1* and *t*, and subsequently undergoes degradation between t and t+1, it receives both a growth and a degradation label.

Table 4. Description of growth transition conditions.

Growth Transition	**Conditions**				**Short Explanation**
	Class at timestep t	**Number of HV nodes in t-1**	**Number of isoHV nodes in t-1**	**Number of nonHV nodes in t-1**	
Expansion	HV	1	0	>=0	HV object grows, no Isolated HV involved.

Eating	HV	1	>=1	>=0	HV object grows by absorbing Isolated HV object(s).
Merging	HV	>1	0	>=0	Multiple HV objects unite.
Eating & Merging	HV	>1	>=1	>=0	Multiple HV objects unite and absorb Isolated HV object(s).
Emerging from one isolated high vegetation	HV	0	1	>=0	New HV object emerges from one Isolated HV object only.
Emerging from multiple isolated high vegetation	HV	0	>1	>=0	New HV object emerges from multiple Isolated HV objects.
Emerging from non-high vegetation	HV	0	0	>=1	New HV object emerges from Non-Isolated HV objects only.

Table 5. Description of degradation transition conditions

Degradation Transition	**Conditions**				**Short Explanation**
	Class at timestep t	**Number of HV nodes in t-1**	**Number of isoHV nodes in t-1**	**Number of nonHV nodes in t-1**	
Shrinking	HV	1	0	>=0	HV object shrinks, remains HV.
Shattering	HV	1	>=1	>=0	HV object fragments into one HV object and Isolated HV object(s).
Splitting	HV	>1	0	>=0	HV object splits into multiple HV objects.
Shattering & Splitting	HV	>1	>=1	>=0	HV object splits into multiple HV objects and Isolated HV object(s).
Shrinking to one isolated high vegetation	HV	0	1	>=0	HV object becomes one Isolated HV object only.
Shrinking to multiple isolated high vegetation	HV	0	>1	>=0	HV object becomes multiple Isolated HV object.
Shrinking to non-high vegetation	HV	0	0	>=1	HV object becomes Non-HV only.

2.4.3 Determining High Vegetation lifecycle paths

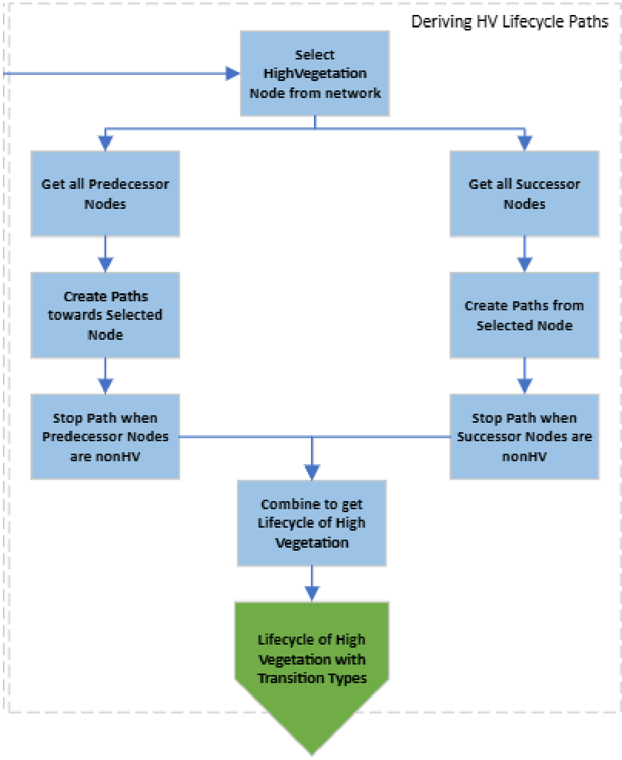


Figure 9. Flowchart with methodological steps for determining the HV lifecycles paths.

To analyze the rise and fall of HV Patches, the lifecycles of those patches need to be evaluated. This is where the temporal network will show its true value. From the temporal network, linked HV Patches can be extracted, and their rise and fall can be traced across multiple timesteps. The python script that is used to derive the lifecycle paths is added In Appendix F.

The lifecycles of HV Patches are reconstructed using depth-first search (DFS) traversal in both temporal directions from each node representing as a HV Patch. First, history paths were reconstructed, tracing all possible backward paths by following incoming edges to predecessor nodes that were also representing HV Patches. The history path stops when no more HV predecessor nodes are found. Then future paths were reconstructed, tracing all possible forward paths by following outgoing edges to successor nodes that were also representing HV Patches. The future path stops when no more HV successor nodes are found. For each HV node, the history and future paths are combined to create complete lifecycle sequences, beginning with the earliest ancestor patch and ending with the most recent descendant. To avoid redundancy, all full lifecycle paths are stored as ordered tuples of (year, object_id) pairs and duplicates are removed. Only unique lifecycle sequences were retained for further analysis. As a result, this method yields a nested list structure, in which each lifecycle is stored as a separate list comprising temporally ordered HV nodes connected across time. With this approach, it is possible to analyze lifecycle durations, frequency distributions, and patterns of persistence or change.

Each node in the network contains attributes describing its growth and degradation transitions, as determined in a prior step. By including these node attributes in the lifecycle sequences, each step in the lifecycle can be associated with its corresponding transition type. In this way, the rise and fall of (riparian) HV patches can be described and researched quantitatively.

As processing the full study area is computationally intensive, a (representative) subset was used for visualizing and analyzing lifecycles

3. Results

3.1 Classifications

3.1.1 Classification statistics

Results of the OBIA classification of the study area for all years are shown in **Figure 10**. The years 1946 to 2002 are classifications classified by Kleinhans et al. (2019). That original dataset has been extended by four new images. Page size figures of these four new classifications are added in **Appendix D1-4**.

In this study we are preliminary interested in large high vegetation patches. For further analysis, Bare soil, low vegetation, main channel and waterbody are taken together as non-high vegetation. Also, all the isolated vegetation patches are taken together as Isolated Vegetation. In **Figure 11**, the total amount of vegetated patches is plotted. Here, we see that the total number of objects varies between different years, ranging from 2000 to 5000 vegetated objects. Another observation is that the amount of HV patches is substantially lower than the amount of isolated vegetation patches.

Instead of analyzing the total counts, it is interesting to investigate the distribution of vegetation objects over the whole area through time. Statistics on the OBIA classification can be analyzed on both the object counts and object areas. **Figure 12** shows the classification statistics for the amount of objects per class or combination of classes for all years. **Figure 13** shows the classification statistics for the area of objects per class or combination of classes for all years.

Outstanding is the difference between count distribution and area distribution. While Isolated High Vegetation is by far the most occurring object in terms of amounts, in terms of area, Isolated High Vegetation only covers a small area. In contrast, although High Vegetation is only a fraction of the total object counts, it consistently covers a much larger area, often two to four times more than its share based on object counts. Lastly, in terms of area, non-High Vegetation covers the largest area, with 50% to 90% of the total area. In terms of counts, non-High Vegetation only accounted for 20% to 50% of all objects.

Given the focus on vegetation structure and landscape dynamics, statistics based on area provide a more meaningful reflection of spatial and temporal patterns than simple object counts. While counts are influenced by segmentation scale and the presence of small, fragmented patches, area-based statistics better capture the spatial extent and ecological dominance of each class. This is also reflected in the consistently higher accuracy values based on area compared to those based on object counts. Therefore, the remainder of this analysis focuses primarily on area distributions rather than object counts.

Temporal patterns in the area distributions (**Figure 13**) reveal several interesting dynamics. In the first years of the study period, Non-High Vegetation clearly dominates. However, from 1992 on, there is a gradual increase in the proportion of High Vegetation area. This increase in High Vegetation area is at the expense of Non-High Vegetation, as the Isolated High Vegetation area stays steadily low. The four new classifications are also added as page-size maps to Appendix C.

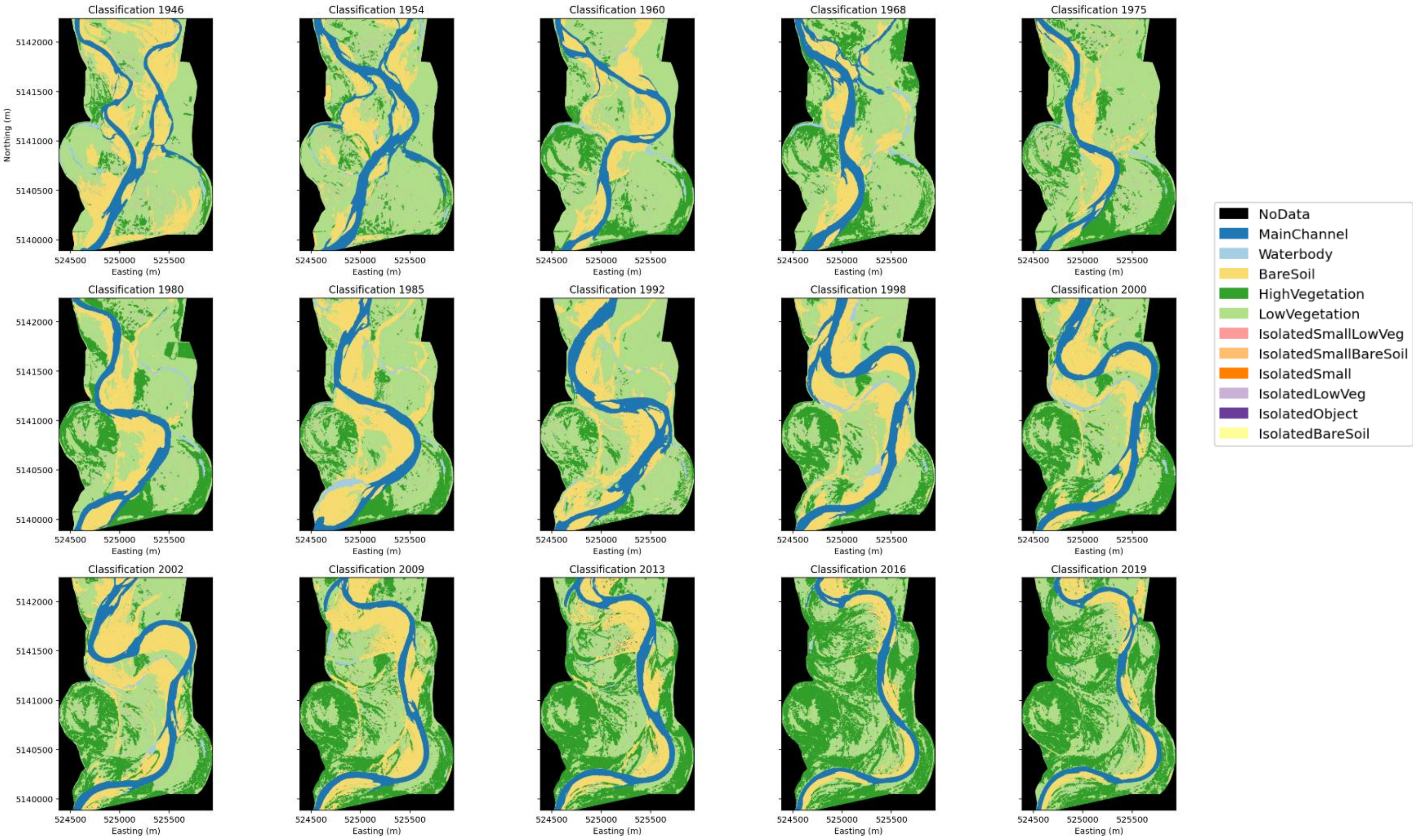


Figure 10. Results OBIA classification.

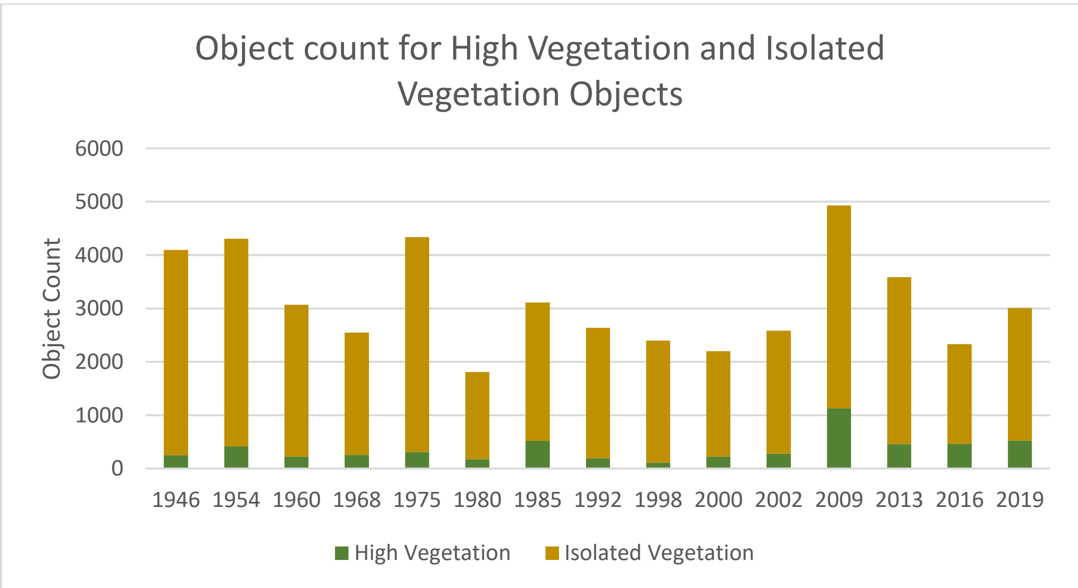


Figure 11. Object count for High Vegetation and isolated Vegetation objects.

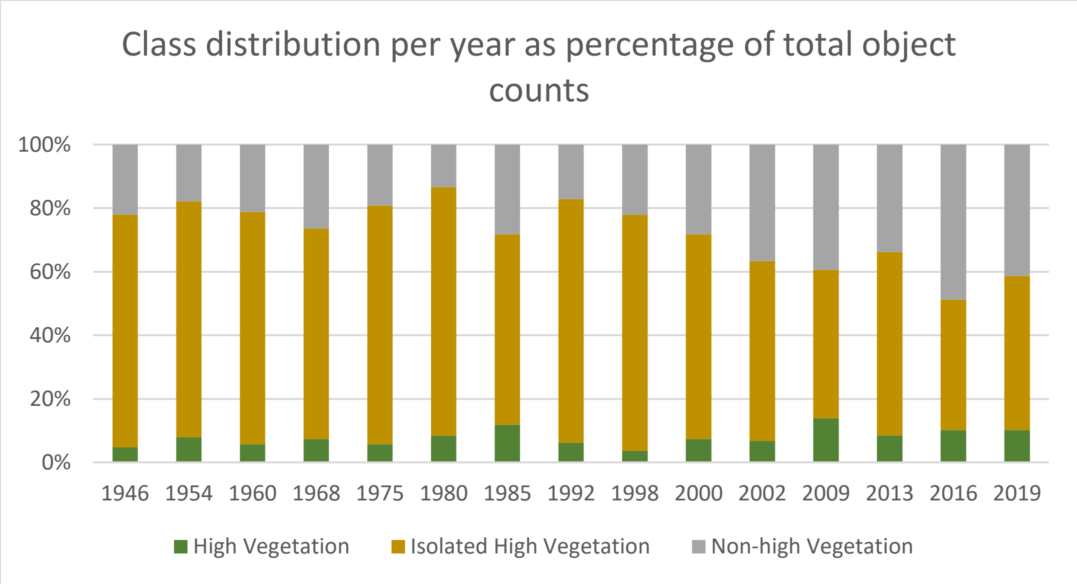


Figure 12. Bar plot showing percentage of object class counts as percentage of total objects.

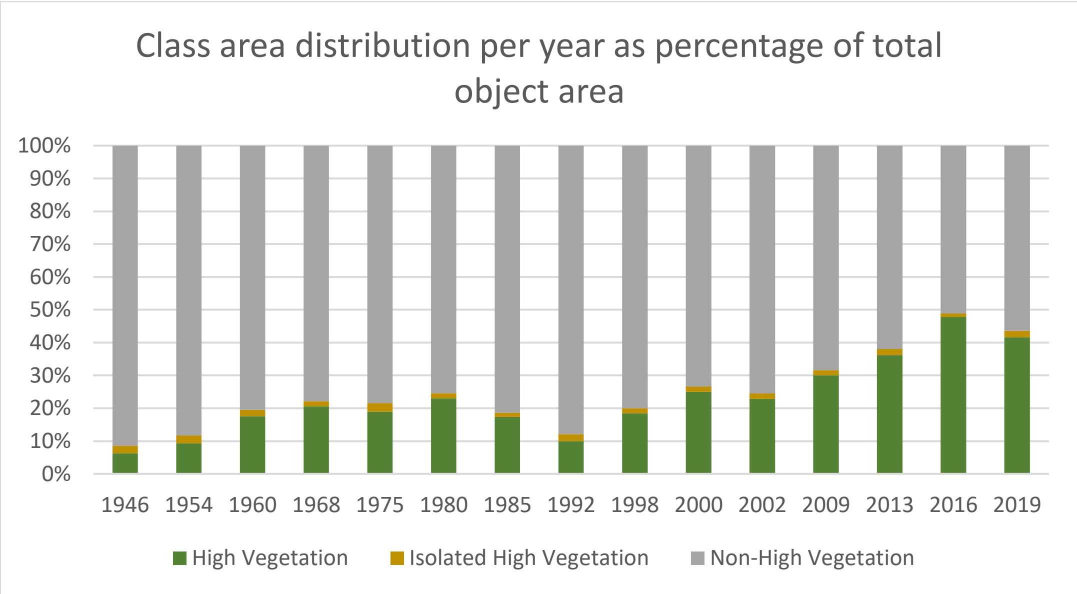


Figure 13. Bar plot showing percentage of object class area as percentage of total area

3.1.2 Accuracy Assessment

To evaluate the performance of the OBIA classifications, a visual accuracy assessment was conducted. The results from the accuracy assessment are shown in **Table 6**. In the **Appendix C1-4**, the full confusion matrices together with the producer's and user's accuracy are attached. Accuracy is high for both object count and areas, with accuracies between 84% and 89% for object count, and steadily around 95% for object area.

One notable observation is that accuracy based on object area is significantly higher than that based on count in every year. This implies that larger objects tend to be classified more accurately than smaller ones. Smaller patches may be more heterogeneous or affected by edge effects, which increases the likelihood of misclassification. In contrast, large and homogeneous patches, particularly those representing major land cover types, are more reliably detected.

As in this research we are mainly interested in the spatial and temporal dynamics of HV patches, special attention was given to the accuracy of this specific class. Detailed producer's and user's accuracies for High Vegetation are presented in **Table 7**, both for object counts and object areas. The results clearly show a substantial improvement in accuracy when area is considered. This pattern is consistent across all years.

The greater discrepancy between count-based and area-based accuracy for High Vegetation compared to the overall accuracy differences suggests that small vegetation patches are more prone to misclassification, while larger High Vegetation patches are consistently well identified. Since the central objective of this research is to track large vegetation structures over time, this high area-based accuracy provides strong support for the reliability of the results presented in the spatial and temporal analyses.

Table 6. Results from accuracy assessment classified maps.

Year	Overall accuracy using object count (%)	Overall accuracy using object areas (%)
2009	89.0	94.6
2013	89.0	94.8
2016	84.0	95.5
2019	87.0	95.3

Table 7. Producer's and User's accuracy of High Vegetation.

		Count		Area	
		Producer's Accuracy (%)	Producer's Accuracy (%)	User's Accuracy (%)	User's Accuracy (%)
Year	2009	76.9	96.1	66.7	81.3
	2013	58.3	97.2	77.8	99.3
	2016	45.0	95.5	75.0	99.2
	2019	58.8	92.4	90.9	88.7

3.2 Temporal Network

The main goal of this study was to introduce a novel framework to track individual vegetation patches over multiple timesteps. To achieve this, a Temporal Network was constructed, in which nodes represent land cover objects from the classified maps, and edges link them across time based on spatial overlap. Each edge indicates a connection between a predecessor and a successor node and includes information on the overlapping area that forms the basis of the link.

Figure 14 shows a 2D representation of the temporal network with nodes and edges for a small area for 4 consecutive years. The size of the nodes is weighed on the size of the object the node represents, and the edges are indicated with the amount of pixels included in the edge. For easier interpretation, the colors of the nodes match the colors of the land cover classes as plotted in **Figure 10**. **Figure 15** shows a 3D representation of the network, where the connected objects are linked in a spatial-temporal setting. As can be seen in the 3D directional network in **Figure 15**, overlapping objects in consecutive years are connected to each other with edges. A Temporal Network like this is a key step in reconstructing lifecycles of individual HV patches, where the rise and fall of individual patches are examined. In **Figure 14** and **Figure 15**, a small area for a short time interval of the last four consecutive time steps is shown.

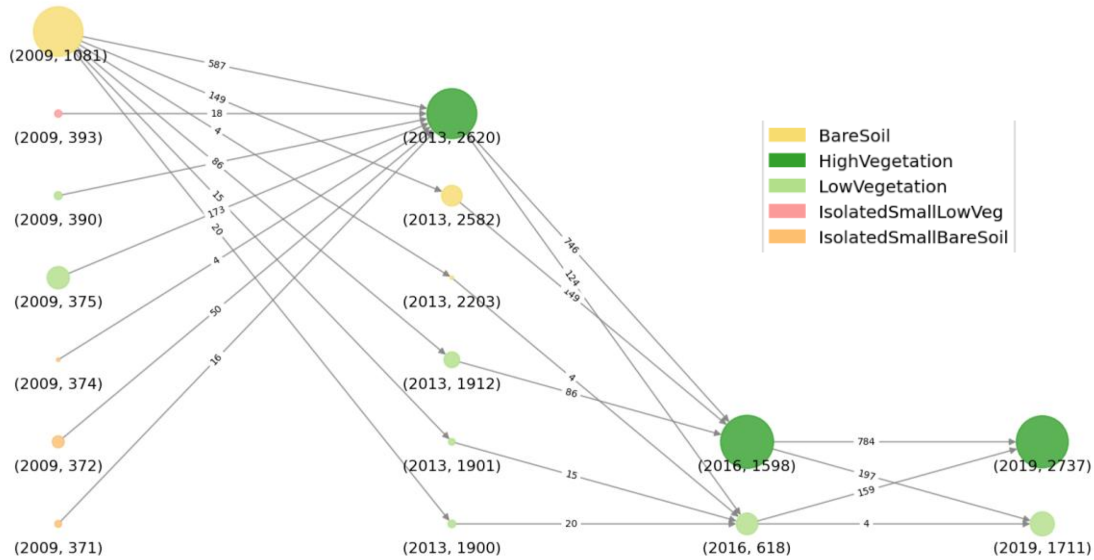


Figure 14. 2D representation of the Temporal Network for a small test area for the years 2009, 2013, 2016, and 2019. The size of the nodes indicates the relative size of the corresponding object, and the color represents its class. In between brackets the nodes' year and object_id is shown. The edges are labeled by the number of pixels included in the edge.

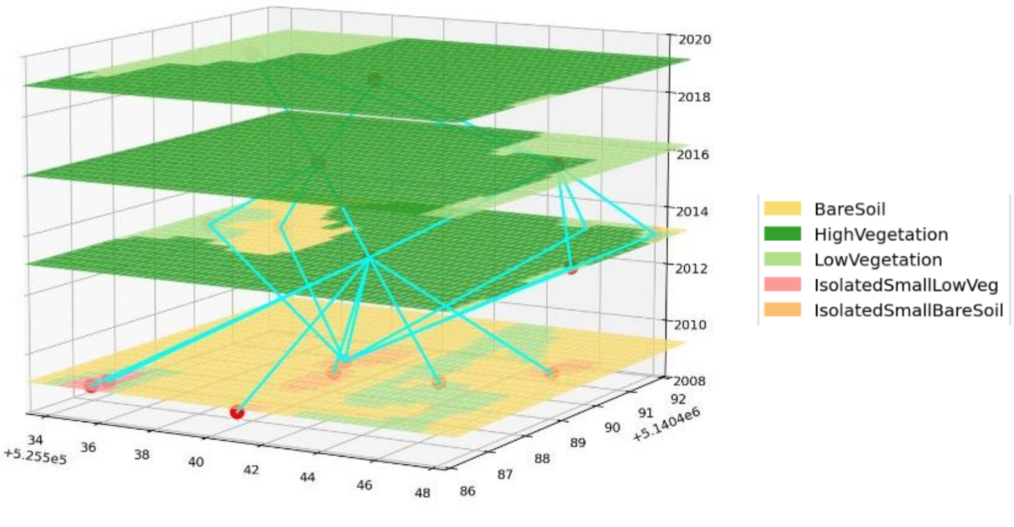


Figure 15. 3D representation of the Temporal Network of Figure 8 with the classified maps of the years 2009, 2013, 2016, and 2019.

3.3 Translation Network to HV lifecycle

The Temporal Network created in this thesis research opens up a possibility to analyze HV patch lifecycles. These lifecycles include linked HV objects over time from first to last appearance. With information on the lifecycles, in combination with the transition types that are attached to all HV nodes, the rise and fall of HV patches can be described.

Ideally, lifecycle reconstruction would be performed for all High Vegetation (HV) patches across the entire study area described in chapter **1.5 Study area**. However, computational limitations arose due to the immense amount of lifecycles created when running the script for the whole area, reaching millions of lifecycles. This resulted in excessive memory and disk usage, making full-area analysis unfeasible. As a result, a smaller subset of the study area was used for lifecycle reconstruction and transition type analysis (**Figure 16**Error! Reference source not found.).

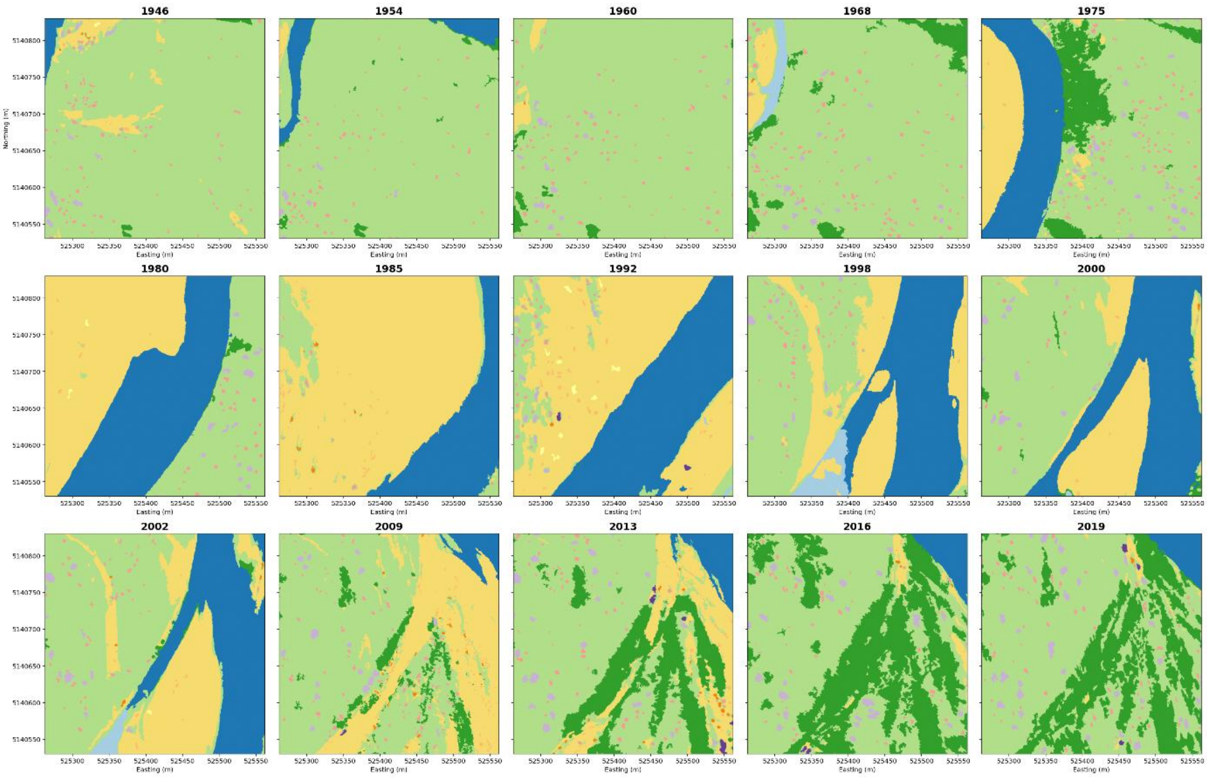


Figure 16. OBIA classification of the smaller subset used for results acquisition.

3.3.1 High Vegetation Lifecycles

To better understand the rise and fall of riparian HV patches, lifecycles were constructed. This was done by identifying temporally connected HV objects across consecutive years. These linked sequences form unique lifecycle trajectories that represent the rise and fall of HV patches throughout the whole time period.

The resulting unique HV lifecycles are visualized in **Figure 17**. Here, in arrows the temporal extent of all HV lifecycles in the study area are shown. While some HV patches persist for long periods, many are short-lived. The most common lifecycle duration is 11 years, but many patches do not reach this age, resulting in an average duration of 9.12 years (**Figure 18**). This shows that although HV patches can persist over longer periods, most experience shorter lifespans.

Also, the years in which the most lifecycles start or end can be identified. A clear observation is the abrupt fall of HV patches after 1975 (**Figure 17**). Following this largescale fall in HV patches, rise of HV patches starts slowly after a roughly 20-year interval. In the first stages of revegetation, lifecycles are short, only extending one to a few years. It is only after 2002 that HV patches survive longer and even remain until 2019, the last year in the data set. Not only do the more recently emerged HV patches tend to survive longer, they are also significantly more abundant than during the periods before. Years in which most HV patches emerge are in the period between 2002 and 2013**.** The increasing number of emerging HV patches, combined with their higher persistency and continued growth after establishing due to stable fluvial conditions results in the overall increase in HV area in the study area. This trend is reflected in **Figure 10** and **Figure 13**, which show both the spatial spread and increasing abundance of HV patches across the study area in the most recent years.

The apparent concentration of HV lifecycle endings in 2019 is a consequence of the temporal extent of the dataset, not an actual peak in patch disappearance. Excluding 2019, HV patch termination occurs at a relatively constant rate, averaging around 10 per year. Only two periods with less disappearance are present (1954-1960 and 1980-1992). The period with barely any HV disappearance between 1980 and 1992 is likely due to the prior rejuvenation phase, where all HV patches already were removed. However, it must also be remarked that interpreting this period is difficult due to the larger time gap between aerial images. On the contrary, the period with below average HV disappearance around 1960 does not follow a rejuvenation phase and time gaps between aerial images are relatively small. This might indicate that the period around 1960 reflects stable environmental factors, resulting in persistent HV patches.

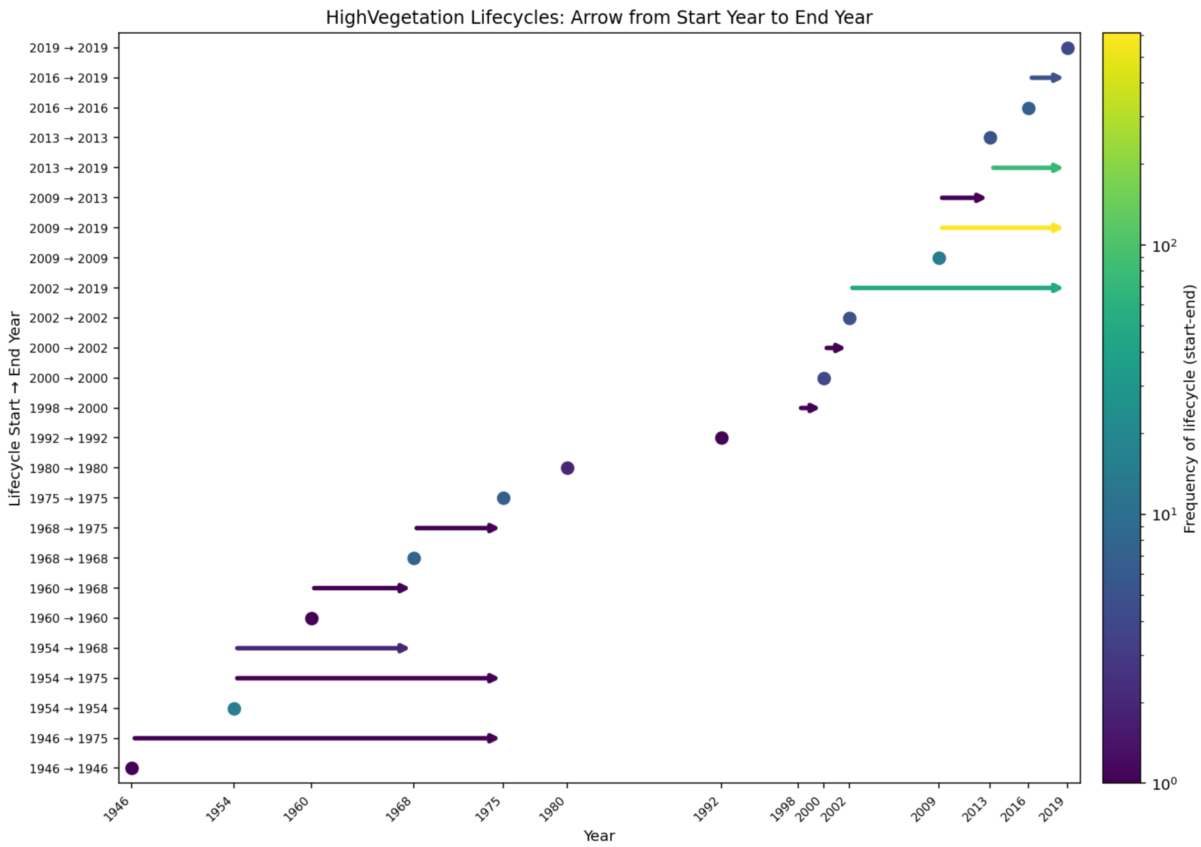


Figure 17. Lifecycles for all HV Vegetation patches (subset 3)

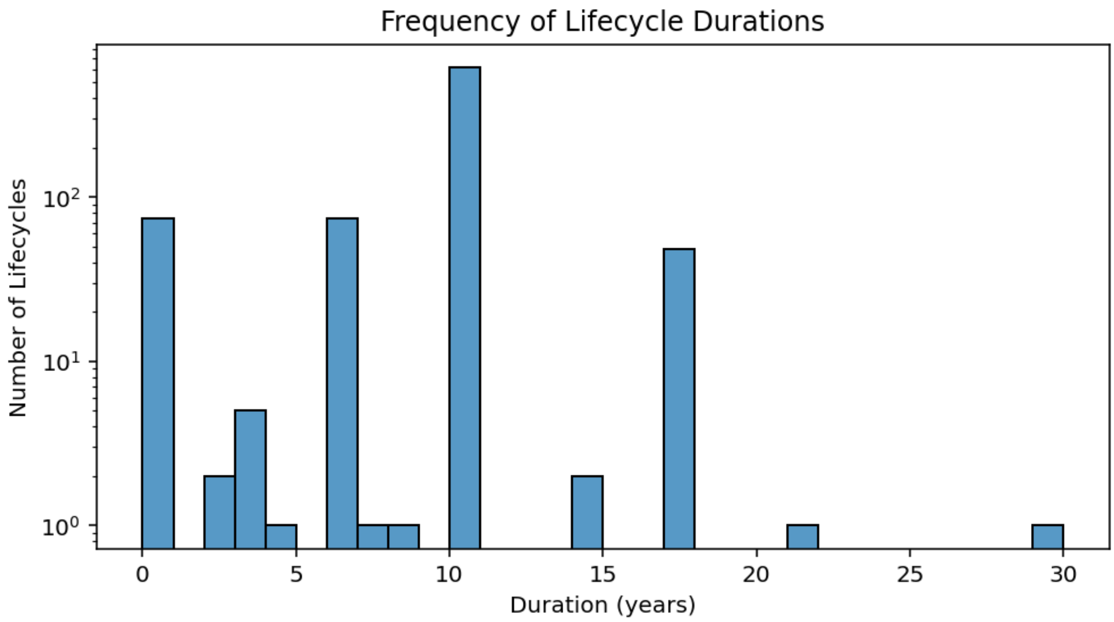


Figure 18. Frequency of lifecycle durations.

3.3.2 Trends in transition types

To better understand the dynamics behind growth and degradation of HV patches, transition types were classified and applied to each node in the temporal network representing a HV object. In **Figure 19**, the total counts of these classified growth and degradation types are shown. Here, it stands out that Emerging from nonHV and Shrinking to nonHV are the most dominant transition types, followed by the other emerging and shrinking transition types. These transition types represent the starting or end nodes of HV lifecycles.

To gain deeper insight in the growth and degradation dynamics within a lifecycle of a HV patch, the transition types are coupled to the nodes within the lifecycles. By doing so, typical growth and degradation patterns are identified and quantified.

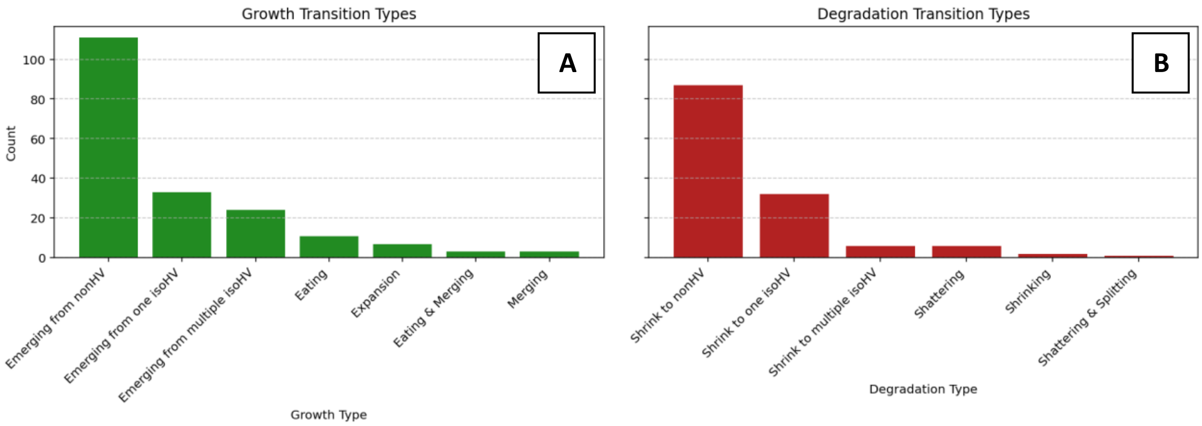


Figure 19. Count of classified growth types (A) and degradation types (B) for all HV patches throughout the study period.

Figure 20, visualizes the distribution of growth and degradation transitions relative to the age of HV patches, expressed in both object count (A and C) and area (B and D). When focusing on the growth transition types, it becomes clear that most HV patches originate from emerging from nonHV objects (**Figure 20A+B**), meaning that HV patches develop from low vegetation and bare soil. There are also some HV patches that emerge from one or multiple isolated HV objects, however, these transitions contribute barely to the total area of emerging HV patches (Figure 20B).

As lifecycles progress, Eating & Merging becomes the dominant growth transition type, indicating that once HV patches have been established, their development is often characterized by absorption of adjacent vegetation patches. Pure Expansion, where a patch grows in area without merging with another HV patch, is relatively rare in earlier stages, but gains more importance in later lifecycle stages. Merging without eating is practically negligible, suggesting that interactions between patches rarely happen in isolation from other nearby changes.

Regarding degradation transition types (Figure 20C+D), it becomes clear that Shrinking to nonHV is the most common, reflecting a direct loss of high vegetation area back into bare or low vegetation states. Shrinking to nonHV may also indicate a transition to water. Yet, in the first 10 to 15 years of lifecycles, transitions like Shattering & Splitting and Shrinking to multiple isoHV also occur frequently. These transitions reflect either fragmentation or partial retreat, rather than complete disappearance, implying that fragmentation can act as a precursor to eventual loss of vegetation. This phased breakdown may be important for identifying early signs of ecological stress or decline.

In Figure 20, all lifecycles are used to calculate the trends in transition types. Because the longest lifecycle in the dataset lasts 29 years (1946-1975), this is the last column in the Figure 20C and Figure 20D. In Figure 20A and Figure 20B, the longest observed lifecycle duration lasts 22 years. Although this lifecycle has its starting node in 1946 and end node in 1975, no growth transitions are given to nodes in 1946. As a results the first recorded growth transition is from 1954, making 22 years the maximum duration of lifecycles including growth transitions. However, as observed in Figure 17 and Figure 18, most lifecycles are significantly shorter. Therefore, as we are interested in the rise and fall of HV patches, it is interesting to see the typical growth and degradation type sequences of HV patches within lifecycles.

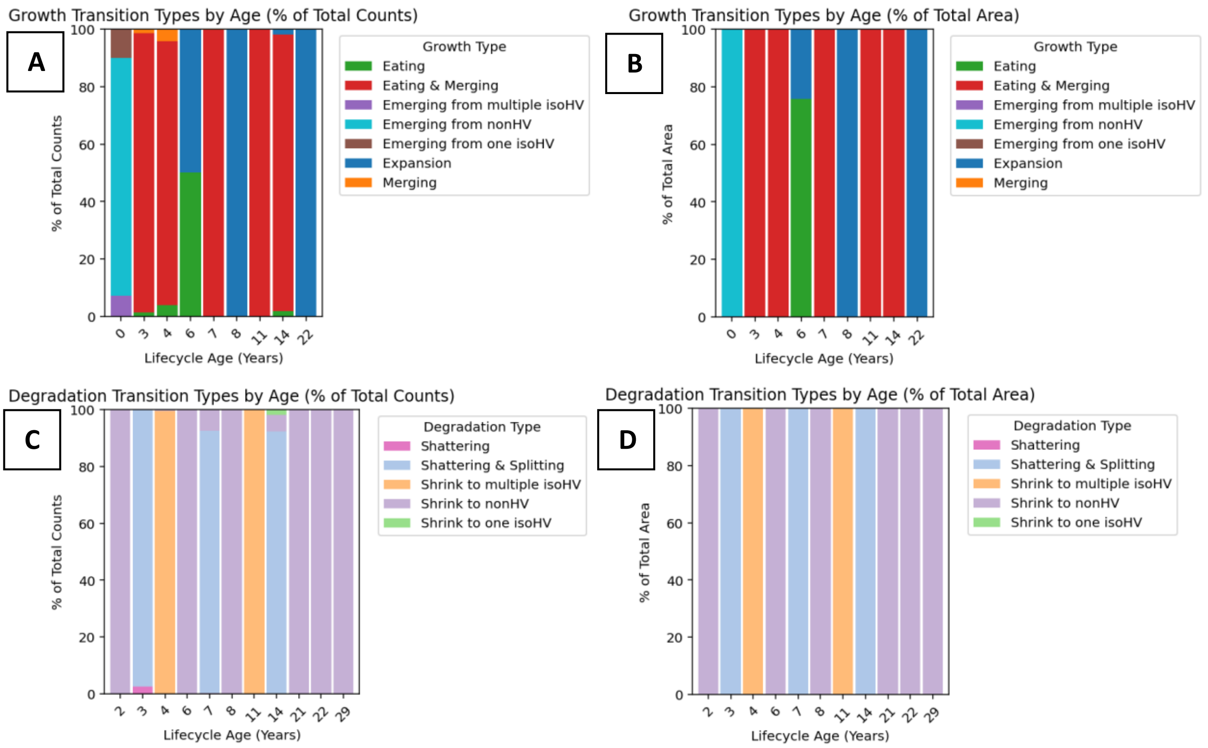


Figure 20. Growth transition types per HV age as percentage of total counts (A) and total area (B). Degradation transition types per HV age as percentage of total counts (C) and total area (D).

Figure 21 and Figure 22 provide further insight into the most typical sequences of growth and degradation. Notably, almost all growth sequences begin with Emerging from nonHV and transition into Eating & Merging. For degradation, the four most frequent sequences end with Shattering and Splitting, a transition type that does not necessarily indicate complete loss of vegetation, but rather a break-up of a formerly cohesive HV patch into multiple fragments, often isolated. This may reflect a transitional phase toward full degradation. When end-of-lifecycle nodes are isolated, Shrinking to nonHV emerges as the most definitive form of vegetation loss. Transition types plotted per year (Figure 23 and Figure 24) reflect similar dynamics: in particular, degradation transition types such as Shrinking to nonHV dominate in years marked by major shifts in river morphology (e.g., post-1975), while growth transition types such as Emerging from nonHV are more common in years following stabilization (e.g., after 2002).

It should be noted, however, that degradation trends are likely influenced by the structure of the dataset. Since 2019 is the final year in the time series, many lifecycles appear to end in that year. As a result, transitions observed between 2016 and 2019 (such as Shattering & Splitting) may reflect partial degradation that would have otherwise continued in subsequent years if more recent data were available. This temporal cutoff could therefore bias the interpretation of degradation trends.

An important point to mention is that in the results where the transition types are connected to HV nodes in the lifecycles, some nodes are labeled incorrectly. In the sequence of transition types shown in Figure 22, Shrinking to nonHV, one isoHV or multiple isoHV are located on places in the lifecycle that are not ending node of a lifecycle. After running the script for a different area than analyzed in the results here, the same type of incorrect labeling appeared to be present in growth transitions, where HV nodes were labeled to Emerging from nonHV, one isoHV or multiple isoHV while not being the starting node of a lifecycle.

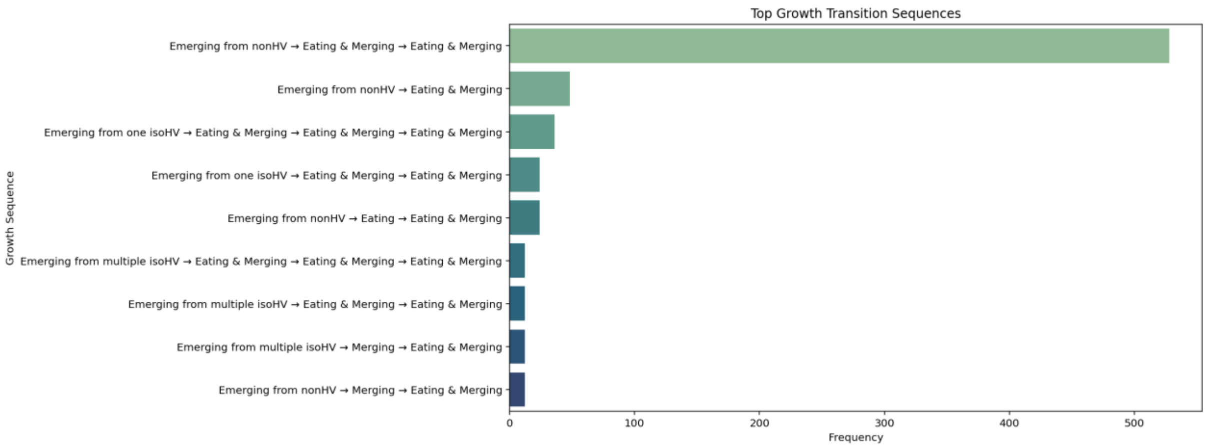


Figure 21. Top 10 most occurring growth sequences.

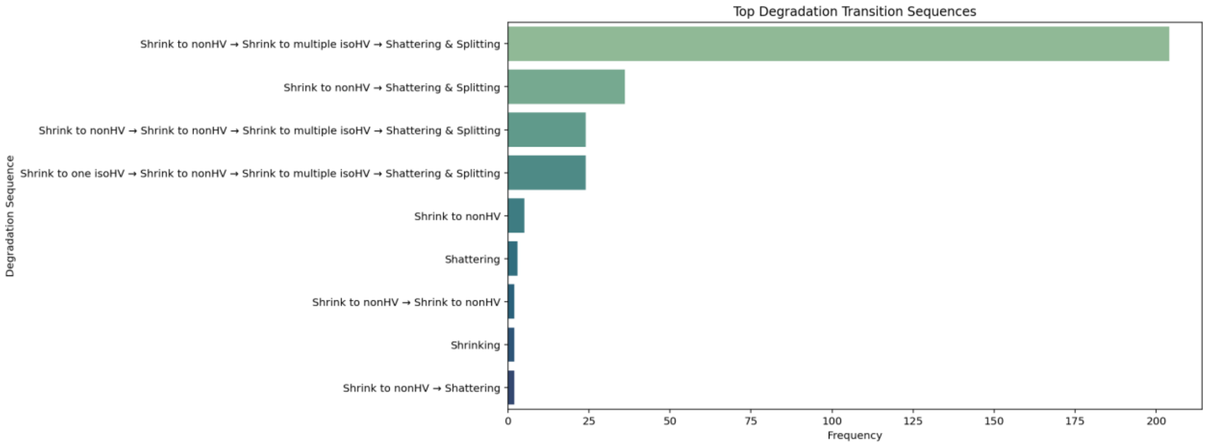


Figure 22. Top 10 most occurring degradation sequences.

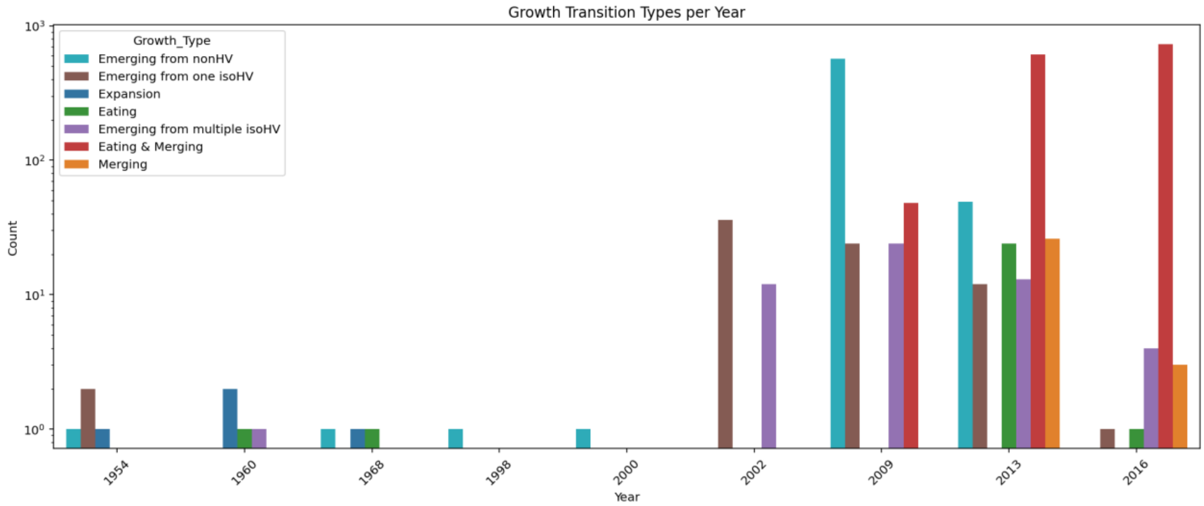


Figure 23. Growth type per year. Only years that include growing HV patches are plotted.

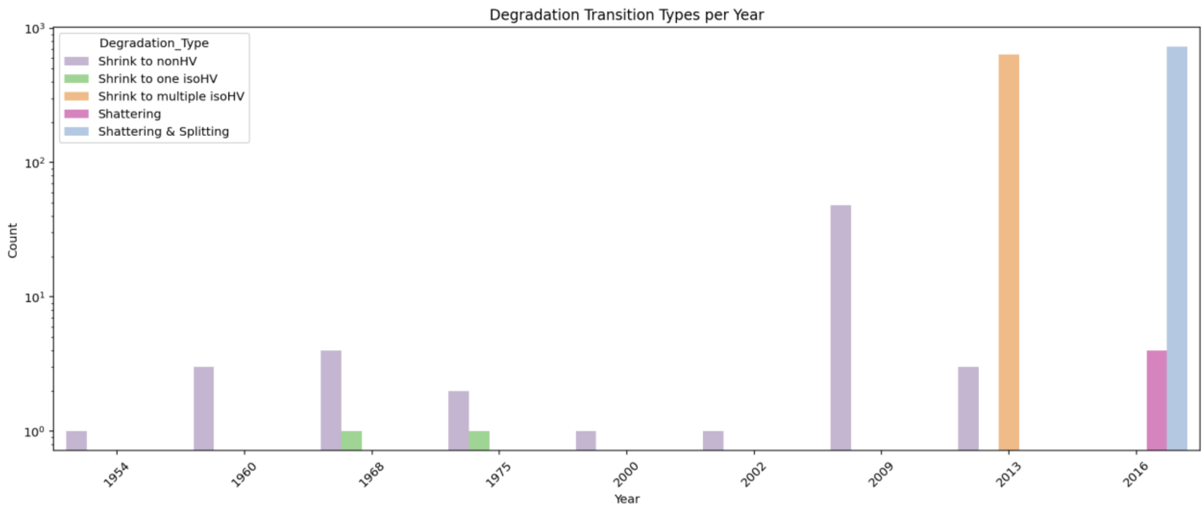


Figure 24. Degradation type per year. Only years that include degrading HV patches are plotted.

3.3 Spatial Patterns in the Rise and Fall of HV

Besides purely taking only the temporal component into account, spatial patterns can also be distinguished regarding HV lifecycles and associated transition types. Spatial patterns in HV patch development reveal strong links with river morphology.

As already mentioned based on **Figure 17**, there are periods in which the rise of HV patches dominate the study area and periods in which the fall of HV patches dominate. A clear rejuvenation period is captured by all HV patches falling after 1975. This period matches a severe shifting period of the Allier River over the (subset) study area (**Figure 10**). The dominant degradation type in that period is Shrinking to nonHV (**Figure 24**), which corresponds to complete removal of HV patches. This removal can thus be linked to erosion due to river inundations.

After the rejuvenation period, the Allier River section in the study area finds a new stable position for a few years around 2000. Between 2002 and 2009 the meander gradually migrated over the area to its current position, whereafter the meander location stayed stable. Rise of HV patches strongly increases in this period of river stability. These HV patches emerge on the former river banks that were formed during the stable period around 2000 (**Figure 10**), mainly by Emerging from nonHV. After initial rise on the former riverbanks, these HV patches expand towards the former riverbed, mainly by Eating & Merging.

4. Discussion

This study aimed to better understand the rise and fall of large, high riparian vegetation patches over time, using a novel framework that combines object-based classification with temporal network analysis. By reconstructing lifecycles and identifying and quantifying transition types, this research provides new insight into how such patches form, persist, and degrade in a dynamic fluvial environment.

4.1 OBIA Classification

The visual accuracy assessment on the newly classified maps showed an accuracy of 84% to 89% for object count, and an accuracy steadily around 95% for object area **Table 6**. This >85% is in line with results from previous studies using OBIA in riparian environments (Osio et al., 2019; Tormos et al., 2012). From this it can be concluded that the OBIA technique used in this research is appropriate for classifying the land cover types in riparian environments. The consistency with earlier research also supports the validity of the stratified random sampling approach used for accuracy assessment, which appears to provide a balanced and representative evaluation of classification performance.

It must be noted that by taking 100 random objects after stratifying, some biases occur. Firstly, as there are significantly more small objects in the stratified sample set, the result of the accuracy assessment using the amount of objects is disproportionately influenced by the classification performance on small patches, which are often more difficult to classify reliably due to their size and spectral ambiguity (Radoux & Bogaert, 2017). Conversely, even though large objects are few, their inclusion in the stratified random sample set can have a significant impact on area-based accuracy, since a single large object contributes much more to the total classified area than a small one. This can lead to area-based accuracies being skewed upwards, especially if these few large objects happen to be correctly classified. The higher values for area-based accuracy across all years suggest that larger objects were more accurately classified than smaller ones. This distinction is particularly relevant given the focus of this study on HV patches.

Another look at the class-specific user's and producer's accuracy values for High Vegetation (**Table 7**) further illustrates the strong contrast between object count- and area-based assessments. Producer's accuracy tells us how many of the actual HV patches were correctly identified by the classification, while user's accuracy shows how often the classified high vegetation objects were actually correct. In all years, producer's accuracy based on area is relatively high, meaning that most of the large HV patches present in the imagery were correctly picked up by the classification. However, the producer's accuracy based on count is much lower, especially in 2016 (only 45%), which shows that relatively small HV objects had a higher chance of being missed. User's accuracy based on count is generally higher than producer's accuracy based on count. For example, in 2019, user's accuracy was 90.9%, while producer's accuracy was only 58.8%. This means that when an object was classified as high vegetation, it was usually correct, but many actual high vegetation objects were not identified. This effect is less visible when using area, where both user's and producer's accuracy are high, confirming that large HV patches were reliably classified. Since this study focuses on the dynamics of large HV patches, the high area-based accuracies give confidence in the results.

While this study does not focus on aggregate temporal land cover change, but rather on the dynamics of individual vegetation patches, the classification results do reveal a broader trend worth noting. From the end of the 20th century onward, a steady increase in HV patches has been observed. This corresponds to the findings of Garófano-Gómez et al. (2017), who linked this increase in High Vegetation to a shift in flow regime of the Allier River. Specifically, the reduction in high and moderate magnitude flood events led to increased channel stability allowing vegetation patches to persist and expand with less frequent disturbance.

4.2 Lifecycle Trends

The combination of Object-Based Image Analysis (OBIA), a temporal network, and lifecycle construction proved valuable for uncovering the rise and fall patterns of riparian vegetation patches. Traditional class-based change detection methods are limited to static comparisons between timesteps, which often miss the spatial continuity and identity of individual patches. By contrast, this approach made it possible to follow vegetation patches through time as unique spatial entities, tracing their rise and fall in detail. The lifecycles allowed the identification of complete HV patch trajectories, from initial establishment to final degradation, providing an entirely new way of interpreting floodplain vegetation dynamics, not as isolated snapshots of change, but as structured sequences of ecological development, with clear beginnings, middles, and ends.

The average lifecycle duration of 9.12 years indicates that most HV patches do not persist over multiple decades but instead experience relatively short-lived periods of stability before degrading. This aligns with the dynamic nature of riparian zones where fluvial disturbances can reset vegetation succession. However, in the more recent decades, more HV lifecycles appear to last longer than the average, indicating that long-term persistence is possible. Although the Allier River is unique in western Europe for its relative undisturbed natural river dynamics (Geerling et al., 2006), anthropogenic impact on river dynamics has become more of influence. Breton et al. (2023) showed that in the last decades, anthropogenic influences on the Rhône river favored more stable riparian environments, resulting in a shift towards more advanced successional vegetation stages. This shift is also observed in the Allier River, where in the last decades the presence of HV patches increased significantly. Not only anthropogenic influences, but also climate change is recognized as a driver for more stable hydrodynamics and increase in forest cover (Garófano-Gómez et al., 2017). Furthermore, the increase in HV patches, in combination with the stable river conditions might be an indication of eco-geomorphological feedback, whereas more vegetation results in more favorable conditions for vegetation growth, which further stabilizes the river location. Vegetation stabilizing the river channel alignment is also found in work by the TU Delft, where the morphodynamic effects of riparian vegetation growth on a re-meandering river is studied (Vargas-Luna et al., 2018).

The timing of lifecycle events reveals the influence of river dynamics on vegetation development. The abrupt disappearance of HV patches after 1975 aligns with known periods of high floods and morphodynamic change in the Allier River (Kleinhans et al., 2019). This widespread disappearance suggests a major rejuvenation event, likely caused by significant channel migration as shown in **Figure 16**, combined with prolonged flooding as documented in Kleinhans et al. (2019). Following this, there is a notable gap in HV emergence, and only from 2002 onwards do new lifecycles start appearing in greater numbers. Patches that emerged in the early 2000s tend to survive longer, some even continuing up to 2019, the final year of observation in this study. This pattern reflects a phase of recovery, likely coinciding with increased geomorphological stability, which provided conditions for recolonization and consolidation of vegetation.

Together, these lifecycle statistics reveal that HV patches exhibit episodic as well as continuous turnover. While the abrupt and widespread disappearance of HV patches after 1975 represents a major loss event, HV patches also degrade more gradually throughout the timeline, with a relatively consistent number of lifecycles ending in most years. This indicates that while large-scale disturbances punctuate the dataset, background degradation, due to local factors such as competition, or small-scale disturbances, also plays an important role in shaping the riparian landscape. These observations align with earlier findings of Mohan & Joseph (2024), confirming that both episodic and background disturbances shape riparian vegetation dynamics.

However, it is important to acknowledge that the results analyzed in this thesis are from a small subset and are therefore not 100% representative for the whole study area. When regarding the whole river stretch of the study area, lifecycles will probably show a more steady pattern without total removal of HV patches as occurring in the subset area. A more steady pattern of emerging and disappearing HV patches would be consistent with previous findings from e.g. Geerling et al. (2006). For future research it would be interesting to verify if this is indeed the case.

4.3 Transition Types within Lifecycles

Having established the timing and duration of HV patch lifecycles, the next step is to understand how these patches change, by analyzing the specific structural dynamics of growth and degradation. This is where the use of transition types adds additional explanatory power.

The combined results from the lifecycles and the transition types show trends that can help to interpret the typical dynamics behind the rise and fall of large, high riparian vegetation (HV) patches. Most HV patches are formed by emerging from nonHV classes, such as bare soil, low vegetation, or water. This dominance of Emerging from nonHV as an initial transition type highlights the role of primary succession as the main pathway for HV patch formation in this system. Rather than expanding from previously existing small vegetation fragments, the majority of the HV patches begins on open substrate, likely made available by previous disturbances or depositional processes. Kleinhans et al. (2019) already mentioned that only a small proportion of the small isolated HV patches merge into larger HV patches. This supports the finding that most HV patches emerge from grassy or open substrate. However, it could be that isolated HV patches are actually present as intermediate step in the appearance of HV patches more often, but that this step is not caught in the results because of the time gap between two consecutive aerial images.

Once established, HV patches tend to grow through Eating & Merging, indicating that HV patch growth is primarily driven by interaction with both nearby small isolated and large HV patches. This mode of growth suggests that vegetation development is primarily governed by the proximity and spatial configuration of other HV patches. This also indicates that positive eco-engineering feedback may be at work, where existing vegetation enhances the potential for further colonization, corresponding to findings in existing literature (Camporeale et al., 2013; Corenblit et al., 2015; Geerling et al., 2006). In later moments during the lifecycle, growth by Expansion starts getting a bigger role (**Figure 20**), suggesting that once a HV patch reaches a certain maturity, they are more likely to expand outward independently, without needing to incorporate other HV patches. This shift may reflect changing ecological conditions: as vegetation establishes, habitats become more favorable for lateral growth, possibly due to increased sediment stability, reduced flow disturbance that support outward expansion (Corenblit & Steiger, 2024; Solari et al., 2016).

On the degradation side, the dominance of Shrinking to nonHV underscores the vulnerability of HV patches to disturbance by river avulsions or competition. However, the presence of fragmentation-based transitions such as Shattering & Splitting in 2013 and 2016 highlights that degradation does not always result in an immediate total disappearance of HV. Instead, degradation may proceed in phases, where patch cohesion is lost gradually (perhaps due to drought stresses resulting in a too low water table (D'Alpaos et al., 2016)) before a patch fully reverts to nonHV. This has important implications for early detection of degradation processes in riparian systems, as fragmentation may serve as a warning sign for future collapse. In the final years of the time series, many HV patches in lifecycles show degradation without fully disappearing. Since the 2016–2019 interval represents the last observable step in many lifecycles, it is difficult to interpret these trends with high confidence, as further transitions beyond 2019 remain unknown. However, like with the growth transition it could be that isolated HV patches are actually present as intermediate step in the disappearance of HV patches more often, but that this step is not caught in the results because of the time gap between two consecutive aerial images.

To summarize, although there is some variability in the lifecycle patterns found in this research, the rise and fall can be typically described as follows: Most frequently, a HV patch emerges from non-isolated HV classes like bare soil or low vegetation, indicating primary ecological succession. Growth is then dominated by a combination of Eating and Merging transitions, through which patches consolidate and expand. After an average lifespan of approximately 10 years, HV patches tend to degrade to non isolated HV, which includes bare soil, low vegetation, and water. River dynamics like meander migration and floods, play a large role in the fall of HV patches. Before complete degradation, HV patches may experience intermediate degradation types, like shattering & splitting. This rise-and-fall trajectory, now observable through the combination of OBIA, temporal networks, and transition logic, reveals the dynamic interplay between ecological succession, spatial structure, and disturbance in riparian vegetation systems.

4.4 Spatial Patterns

Linking the spatial position of growing or degrading HV patches could provide a more detailed understanding of the environmental conditions influencing the rise and fall of riparian vegetation. For example, in this thesis research it was already mentioned that HV patches appear to emerge initially on former riverbanks. Kleinhans et al. (2019) analyzed several spatial drivers to understand the dynamics of isolated small riparian vegetation patches. The spatial drivers that appeared to be of influence were distance to water, distance to HV patches, and distance to isolated HV patches. It would be interesting for further research to analyze these spatial drivers in regard to the rise and fall of large riparian vegetation patches. It can be hypothesized that proximity to the active river channel is likely to play a key role, as areas close to the channel are more frequently exposed to disturbances. These dynamic zones may show higher rates of degradation transition types, such as shrinking or fragmentation, but may also facilitate emergence and growth of HV patches due to higher moisture availability or nutrient input. In contrast, patches located on older fluvial surfaces, such as former bars, may be associated with higher stability or serve as favorable sites for the emergence of new HV patches during periods of low disturbance.

4.5 Limitations of this study

The framework proposed in this study has proven valuable for monitoring the rise and fall of large, high riparian vegetation patches. However, several limitations should be acknowledged and taken into account when interpreting the results.

Firstly, the proposed temporal network approach needs to deal with uncertainties in the border regions of the study area. At the borders, the size of objects will always be truncated to a certain degree. The impact of this limitation depends on the overall size of the study area. The larger the study area, the smaller the proportion of border objects, and thus the less significant the effect. For the relatively small subset area used in this study, the effect might be larger, as many large objects are partially cut off, meaning their true development over time, reflected by predecessors and successors, is not fully captured in the analysis.

Regarding the creation of the lifecycles, it is important to note that the lifecycle durations are constrained by the temporal resolution of the dataset. The time gaps between years are not constant over the whole study period. Therefore, the start and end years of lifecycles do have an uncertainty margin of a few years. Periods with a longer time interval are expected to have larger changes as more disturbances may have occurred.

Also, incorrect transition type labeling was identified regarding emerging and disappearing HV patches. This error in assigning transition types to the HV nodes can theoretically arise due to a few factors in the script. First, the correctness of the growth and degradation transition conditions were considered as potential source for the error. However, according to how the transition types are defined in the methodology (**Table 4** & **Table 5**), a HV node can only be classified as Emerging from nonHV, one isoHV, or multiple isoHV when there are no HV predecessors. Similarly, a HV node can only be classified as Shrinking to nonHV, one isoHV, or multiple isoHV when there are no HV successors. In the majority of cases, the transition classification aligns with these criteria. Nevertheless, when analyzing the sequence of growth and degradation transition types within lifecycles, instances were observed where HV nodes with predecessor and successor HV nodes were labeled as emerging or disappearing.

These cases indicate that, under certain conditions, the correct identification of predecessor and successor HV nodes may not be functioning as intended. This is likely due to a bug in the script that is not yet identified. Several potential causes were investigated. The implemented condition logic was reviewed and found to be consistent with the transition definitions. Next, the datatypes may be incorrect for the logic used in the script or the datatype changes unintentionally somewhere in the process, e.g. integers and floats are confused. Additionally, the procedures for reading and linking predecessor and successor nodes were considered as possible sources of error.

Despite thorough testing and review, no definitive cause for the incorrect labeling was identified within the available timeframe. While most transition assignments appear accurate and the framework generally performs as expected, the observed inconsistencies introduce a degree of uncertainty when coupling transition types to lifecycle trajectories.

Lastly, due to computational limitations, only a representative subset of the full study area could be analyzed for lifecycle reconstruction, which may limit the spatial generalizability of some findings.

These limitations are primarily caused by the current handling of lifecycle paths in the script, which stores them as large in-memory lists. This approach leads to excessive memory consumption and disk usage, especially when millions of lifecycles are generated. The resulting files can reach sizes of several hundred gigabytes. Future improvements to the script should focus on optimizing memory management and file handling to enable full-scale analysis of large spatial datasets.

Despite these limitations, the core findings, particularly those related to dominant lifecycles, remain robust and offer valuable insights into riparian vegetation dynamics.

4.6 Future research directions

While this study focused on reconstructing and classifying patch trajectories, it also opens the door to predictive modeling of vegetation dynamics. If HV patch lifecycles follow recognizable sequences of transitions, then it may be possible to anticipate their future development or decline. To move toward such predictive capacity, further work is needed to link transition types to spatial variables such as distance to the river channel, topographic elevation, groundwater availability, or proximity to other vegetation. These environmental drivers, combined with transition logic, could offer a pathway toward spatially eco-morphodynamic modeling for riparian vegetation development.

Beyond riparian systems, the methodology used in this study is applicable to other vegetation types and environmental contexts. The network-based approach for multi-temporal object tracking can be adapted to study spatio-temporal changes in other fields. Consider study areas as forest monitoring, mountain biogeomorphology, dune colonization, or maybe even non vegetation related studies, wherever objects evolve over space and time.

Altogether, this study demonstrates the potential of a temporal network approach to extract process-oriented insights from time series of classified imagery. Its broader relevance lies in offering a framework for analyzing spatially structured ecological change, while also laying the foundation for future research that can not only track change, but also predict

5. Conclusion

This study introduced a novel object-based temporal network approach to track the rise and fall of large, high riparian vegetation (HV) patches over time. In this study, individual patch lifecycles were reconstructed and analyzed by linking classified land cover objects across 15 time steps from 1946 to 2019. The method successfully reconstructed the rise and fall of HV patches, providing detailed insights into their duration, transition types, and spatial behavior, and filled a gap in how patch-scale vegetation dynamics are studied in fluvial landscapes.

The results show that most large HV patches follow a rise–growth–fall trajectory, typically lasting around 10 years. Growth most often begins through emergence from bare soil or low vegetation, while further development occurs through merging and eating of nearby patches. Degradation usually occurs through direct disappearance to non high vegetation patches, like bare soil, low vegetation or water, while an intermediate step of patches falling apart to smaller HV patches is also observed.

A major turning point occurred around 1975, when many HV patches disappeared following a period of intense meander migration. In contrast, the period after 2000 saw a steady increase in HV patch frequency and persistency, coinciding with reduced flood activity and increased channel stability. These patterns suggest that river avulsions and meander migration contribute to HV patch loss, whereas stable river conditions promote the emergence, growth and persistence of HV patches.

These findings demonstrate the value of analyzing vegetation at the object level, revealing spatial and temporal dynamics that are hidden in traditional class-based change analyses. By focusing on individual lifecycles and transitions, this study provides a more nuanced understanding of how riparian vegetation responds to environmental and geomorphological change over time.

By shifting focus from areal land cover change to the dynamics of individual patches, the framework introduced in this study provides new insight into the spatial and temporal processes that shape riparian vegetation. It lays the basis for future research into environmental drivers and offers a method that can also be applied to other dynamic landscapes.

6. Reference

Addink, E. A., Van Coillie, F. M. B., & De Jong, S. M. (2012). Introduction to the GEOBIA 2010 special issue: From pixels to geographic objects in remote sensing image analysis. *International Journal of Applied Earth Observation and Geoinformation*, *15*, 1–6. https://doi.org/10.1016/j.jag.2011.12.001

Bätz, N., Colombini, P., Cherubini, P., & Lane, S. N. (2016). Groundwater controls on biogeomorphic succession and river channel morphodynamics. *Journal of Geophysical Research: Earth Surface*, *121*(10), 1763–1785. https://doi.org/10.1002/2016JF004009

Bertagni, M. B., Perona, P., & Camporeale, C. (2018). Parametric transitions between bare and vegetated states in water-driven patterns. *Proceedings of the National Academy of Sciences*, *115*(32), 8125–8130. https://doi.org/10.1073/pnas.1721765115

Bertoldi, W., Drake, N. A., & Gurnell, A. M. (2011). Interactions between river flows and colonizing vegetation on a braided river: exploring spatial and temporal dynamics in riparian vegetation cover using satellite data. *Earth Surface Processes and Landforms*, *36*(11), 1474–1486. https://doi.org/10.1002/esp.2166

Breton, V., Girel, J., & Janssen, P. (2023). Long-term changes in the riparian vegetation of a large, highly anthropized river: Towards less hygrophilous and more competitive communities. *Ecological Indicators*, *155*, 111015. https://doi.org/10.1016/j.ecolind.2023.111015

Camporeale, C., Perucca, E., Ridolfi, L., & Gurnell, A. M. (2013). Modeling the Interactions betweeen River Morphodynamics and Riparian Vegetation. *Reviews of Geophysics*, *51*(3), 379–414. https://doi.org/10.1002/rog.20014

Centre Régional Auvergne-Rhône-Alpes de l'Information Géographique (CRAIG). (2025). *Ortophotographie - Département d'Allier*. https://ids.craig.fr/geocat/srv/dut/catalog.search#/search?any=orthophotographie

Chuman, T., Lipský, Z., & Matějček, T. (2006). Succession of vegetation in alluvial floodplains after extreme floods. *Geografie*, *111*(3), 314–325. https://doi.org/10.37040/geografie2006111030314

Congalton, R. G. (1991). A review of assessing the accuracy of classifications of remotely sensed data. *Remote Sensing of Environment*, *37*(1), 35–46. https://doi.org/10.1016/0034-4257(91)90048-B

Corenblit, D., Davies, N. S., Steiger, J., Gibling, M. R., & Bornette, G. (2015). Considering river structure and stability in the light of evolution: feedbacks between riparian vegetation and hydrogeomorphology. *Earth Surface Processes and Landforms*, *40*(2), 189–207. https://doi.org/10.1002/esp.3643

Corenblit, D., & Steiger, J. (2024). Fluvial biogeomorphological feedbacks from plant traits to the landscape: Lessons from selected French rivers in line with A.M. Gurnell's influential contribution. *River Research and Applications*, *40*(6), 1020–1023. https://doi.org/10.1002/rra.4307

D'Alpaos, A., Toffolon, M., & Camporeale, C. (2016). Ecogeomorphological feedbacks of water fluxes, sediment transport and vegetation dynamics in rivers and estuaries. *Advances in Water Resources*, *93*, 151–155. https://doi.org/10.1016/j.advwatres.2016.05.019

Douda, J., Čejková, A., Douda, K., & Kochánková, J. (2009). Development of alder carr after the abandonment of wet grasslands during the last 70 years. *Annals of Forest Science*, *66*(7), 712–712. https://doi.org/10.1051/forest/2009065

Garófano-Gómez, V., Metz, M., Egger, G., Díaz-Redondo, M., Hortobágyi, B., Geerling, G., Corenblit, D., & Steiger, J. (2017). Vegetation succession processes and fluvial dynamics of a mobile temperate riparian ecosystem: the lower Allier River (France). *Géomorphologie : Relief, Processus, Environnement*, *23*(3), 187–202. https://doi.org/10.4000/geomorphologie.11805

Geerling, G. W., Ragas, A. M. J., Leuven, R. S. E. W., van den Berg, J. H., Breedveld, M., Liefhebber, D., & Smits, A. J. M. (2006). Succession and rejuvenation in floodplains along the River Allier (France). *Hydrobiologia*, *565*(1), 71–86. https://doi.org/10.1007/s10750-005-1906-6

Grabowski, R. C., Vercruysse, K., Holman, I., Azhoni, A., Bala, B., Shankar, V., Beale, J., Mukate, S., Poddar, A., Peng, J., & Meersmans, J. (2022). The land–river interface: a conceptual framework of environmental process interactions to support sustainable development. *Sustainability Science*, *17*(4), 1677–1693. https://doi.org/10.1007/s11625-022-01150-x

Gurnell, A. (2014). Plants as river system engineers. *Earth Surface Processes and Landforms*, *39*(1), 4–25. https://doi.org/10.1002/esp.3397

Hubble, T. C. T., Docker, B. B., & Rutherfurd, I. D. (2010). The role of riparian trees in maintaining riverbank stability: A review of Australian experience and practice. *Ecological Engineering*, *36*(3), 292–304. https://doi.org/10.1016/j.ecoleng.2009.04.006

Kivela, M., Arenas, A., Barthelemy, M., Gleeson, J. P., Moreno, Y., & Porter, M. A. (2014). Multilayer networks. *Journal of Complex Networks*, *2*(3), 203–271. https://doi.org/10.1093/comnet/cnu016

Kleinhans, M., Douma, H., & Addink, E. A. (2019). Fate of pioneering vegetation patches in a dynamic meandering river. *Earth Surface Processes and Landforms*, *44*(8), 1618–1632. https://doi.org/10.1002/esp.4596

Kui, L., Stella, J. C., Lightbody, A., & Wilcox, A. C. (2014). Ecogeomorphic feedbacks and flood loss of riparian tree seedlings in meandering channel experiments. *Water Resources Research*, *50*(12), 9366–9384. https://doi.org/10.1002/2014WR015719

Osio, A., Lefèvre, S., Ogao, P., & Ayugi, S. (2019). OBIA-based monitoring of riparian vegetation applied to the identification of degraded Acacia Xanthophloea along Lake Nakuru, Kenya. *GEOBIA 2018- From Pixels to Ecosystems and Global Sustainability , Centre d'Etudes Spatiales de La BIOsphère (CESBIO); Office National d'études et de Recherches Aérospatiales (ONERA); Espace Pour Le Développement (ESPACE DEV); Société T.E.T.I.S*, 1–14.

Radoux, J., & Bogaert, P. (2017). Good Practices for Object-Based Accuracy Assessment. *Remote Sensing*, *9*(7), 646. https://doi.org/10.3390/rs9070646

Riis, T., Kelly-Quinn, M., Aguiar, F. C., Manolaki, P., Bruno, D., Bejarano, M. D., Clerici, N., Fernandes, M. R., Franco, J. C., Pettit, N., Portela, A. P., Tammeorg, O., Tammeorg, P., Rodríguez-González, P.

M., & Dufour, S. (2020). Global overview of ecosystem services provided by riparian vegetation. *BioScience*, *70*(6), 501–514. https://doi.org/10.1093/biosci/biaa041

Rutherford, I. D., Anderson, B., & Ladson, A. (2007). Managing the effects of riparian vegetation on flooding. In *Principles for riparian lands management* (pp. 63–84). Land and Water Australia.

Mohan, N. S., & Joseph, S. (2024). Disturbances on Riparian Vegetation: A Comprehensive Review. *International Journal of Research and Review*, *11*(4), 200–208. https://doi.org/10.52403/ijrr.20240422

Solari, L., Van Oorschot, M., Belletti, B., Hendriks, D., Rinaldi, M., & Vargas-Luna, A. (2016). Advances on modelling riparian vegetation—hydromorphology interactions. *River Research and Applications*, *32*(2), 164–178. https://doi.org/10.1002/rra.2910

Tormos, T., Kosuth, P., Durrieu, S., Dupuy, S., Villeneuve, B., & Wasson, J. G. (2012). Object-based image analysis for operational fine-scale regional mapping of land cover within river corridors from multispectral imagery and thematic data. *International Journal of Remote Sensing*, *33*(14), 4603–4633. https://doi.org/10.1080/01431161.2011.637093

Trimble. (2025). *eCognition Developer 9*. Trimble Germany GmbH, Arnulfstrasse 126, D80636 Munich, Germany. https://geospatial.trimble.com/en/products/software/trimble-ecognition

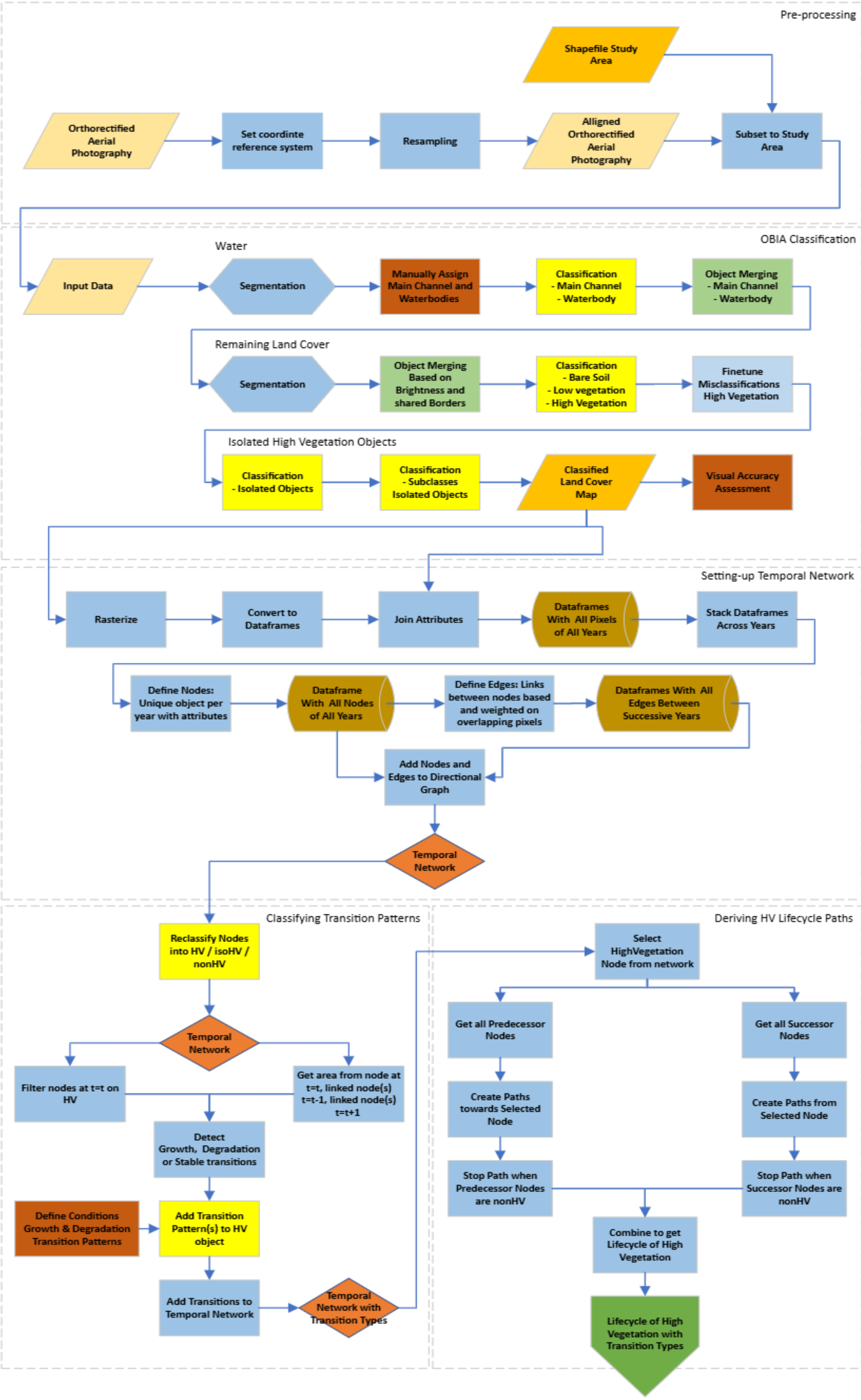
van Dijk, W. M., Schuurman, F., van de Lageweg, W. I., & Kleinhans, M. G. (2014). Bifurcation instability and chute cutoff development in meandering gravel-bed rivers. *Geomorphology*, *213*, 277–291.

van Hemel, G. (2019). *The rise and fall of riparian vegetation patches* [Bachelor Thesis Earth Sciences]. Utrecht University.

Vargas-Luna, A., Crosato, A., Anders, N., Hoitink, A. J. F., Keesstra, S. D., & Uijttewaal, W. S. J. (2018). Morphodynamic effects of riparian vegetation growth after stream restoration. *Earth Surface Processes and Landforms*, *43*(8), 1591–1607. https://doi.org/10.1002/esp.4338

Appendices

Appendix A. Flowchart Methodologies



Appendix B. Finetune Rules OBIA

2009:

- HighVegetation with Rel. border to LowVegetation > 0.4 and Length/Width > 4 and Distance to BareSoil < 100 Pxl at Main Level: LowVegetation
- LowVegetation with Rel. border to HighVegetation > 0.25 and GLCM Contrast (all dir.) > 320 and Brightness < 140 at Main Level: HighVegetation
- 50x: LowVegetation with Rel. border to MainChannel >= 0.4 and Length/Width > 6 at Main Level: MainChannel

2013:

- unclassified with Brightness > 107 and GLCM Contrast (all dir.) < 430 and GLCM Entropy (all dir.) < 8 at Main Level: LowVegetation
- HighVegetation with Rel. border to LowVegetation > 0.8 and Length/Width > 5 at Main Level: LowVegetation
- 3x: LowVegetation with Rel. border to HighVegetation > 0.5 and GLCM Contrast (all dir.) > 133 at Main Level: HighVegetation
- 50x: LowVegetation with Rel. border to MainChannel >= 0.4 and Length/Width > 6 at Main Level: MainChannel

2016:

- unclassified with Brightness > 82.9 and GLCM Contrast (all dir.) < 93 at Main Level: LowVegetation
- HighVegetation with Rel. border to LowVegetation > 0.4 and Length/Width > 2 and Distance to MainChannel < 200 Pxl at Main Level: LowVegetation
- 3x: LowVegetation with Rel. border to HighVegetation > 0.25 and GLCM Contrast (all dir.) > 50 and Brightness < 105 at Main Level: HighVegetation
- 50x: LowVegetation with Rel. border to MainChannel >= 0.4 and Length/Width > 6 at Main Level: MainChannel

2019:

- HighVegetation with Rel. border to LowVegetation > 0.4 and Length/Width > 2 and Distance to MainChannel < 200 Pxl at Main Level: LowVegetation
- 3x: LowVegetation with Rel. border to HighVegetation > 0.25 and GLCM Contrast (all dir.) > 60 and Brightness < 120 at Main Level: HighVegetation
- 50x: LowVegetation with Rel. border to MainChannel >= 0.4 and Length/Width > 6 at Main Level: MainChannel

Appendix C1. Classification Accuracy Assessment 2009

2009 count		**Observed**						
		BareSoil	HighVegetation	LowVegetation	IsolatedSmall	IsolatedSmallBareSoil	IsolatedSmallLowVeg	IsolatedLowVeg
Predicted	BareSoil	**15**						
	HighVegetation		**10**	**2**	**1**	**1**	**1**	
	LowVegetation	**1**	**3**	**24**				
	IsolatedSmall				**2**	**1**		
	IsolatedSmallBareSoil					**16**		
	IsolatedSmallLowVeg				**1**		**19**	
	IsolatedLowVeg							**3**
Overall Accuracy (%)		**89**						

2009 area (m2)		**Observed**						
		BareSoil	HighVegetation	LowVegetation	IsolatedSmall	IsolatedSmallBareSoil	IsolatedSmallLowVeg	IsolatedLowVeg
Predicted	BareSoil	**508.6**						
	HighVegetation		**761.9**	**143.6**	**21.7**	**3.5**	**7.1**	
	LowVegetation	**9.0**	**30.9**	**2273.0**				
	IsolatedSmall				**16.9**	**6.5**		
	IsolatedSmallBareSoil					**73.7**		
	IsolatedSmallLowVeg				**4.4**		**185.6**	
	IsolatedLowVeg							**127.1**
Overall Accuracy (%)		**94.6**						

2009 count	Producer's Accuracy (%)	User's Accuracy (%)
BareSoil	93.8	100.0
HighVegetation	76.9	66.7
LowVegetation	92.3	85.7
IsolatedSmall	50.0	66.7
IsolatedSmallBareSoil	88.9	100.0
IsolatedSmallLowVeg	95.0	95.0
IsolatedLowVeg	100.0	100.0

2009 area (m2)	Producer's Accuracy (%)	User's Accuracy (%)
BareSoil	98.3	100.0
HighVegetation	96.1	81.2
LowVegetation	94.1	98.3
IsolatedSmall	39.3	72.3
IsolatedSmallBareSoil	88.1	100.0
IsolatedSmallLowVeg	96.3	97.7
IsolatedLowVeg	100.0	100.0

Appendix C2. Classification Accuracy Assessment 2013

2013 count		Observed							
		BareSoil	HighVegetation	LowVegetation	IsolatedSmall	IsolatedSmallBareSoil	IsolatedSmallLowVeg	IsolatedLowVeg	Waterbody
Predicted	BareSoil	8							
	HighVegetation		7	2					
	LowVegetation	1	4	18					1
	IsolatedSmall				1	2			
	IsolatedSmallBareSoil					26			
	IsolatedSmallLowVeg		1				18		
	IsolatedLowVeg							8	
	Waterbody								3
Overall Accuracy (%)		89							

2013 area (m2)		Observed							
		BareSoil	HighVegetation	LowVegetation	IsolatedSmall	IsolatedSmallBareSoil	IsolatedSmallLowVeg	IsolatedLowVeg	Waterbody
Predicted	BareSoil	1773.4							
	HighVegetation		5098.3	37.8					
	LowVegetation	1.6	137.3	1188.7					3.7
	IsolatedSmall				20.5	24.5			
	IsolatedSmallBareSoil					171.7			
	IsolatedSmallLowVeg		9.9				213.8		
	IsolatedLowVeg							532.5	
	Waterbody								438.6
Overall Accuracy (%)		97.8							

2013 count	Producer's Accuracy (%)	User's Accuracy (%)
BareSoil	88.9	100
HighVegetation	58.33	77.78
LowVegetation	90.00	75.00
IsolatedSmall	100.00	33.33
IsolatedSmallBareSoil	92.86	100.00
IsolatedSmallLowVeg	100.00	94.74
IsolatedLowVeg	100	100
Waterbody	75	100

2013 area (m2)	Producer's Accuracy (%)	User's Accuracy (%)
BareSoil	99.9	100.0
HighVegetation	97.2	99.3
LowVegetation	96.9	89.3
IsolatedSmall	100.0	45.6
IsolatedSmallBareSoil	87.5	100.0
IsolatedSmallLowVeg	100.0	95.6
IsolatedLowVeg	100.0	100.0
Waterbody	99.2	100.0

Appendix C3. Classification Accuracy Assessment 2016

2016 count		Observed						
		BareSoil	HighVegetation	LowVegetation	IsolatedSmall	IsolatedSmallBareSoil	IsolatedSmallLowVeg	IsolatedLowVeg
Predicted	BareSoil	5						
	HighVegetation		9	3				
	LowVegetation		11	33				
	IsolatedSmall						1	
	IsolatedSmallBareSoil					7		
	IsolatedSmallLowVeg			1			23	
	IsolatedLowVeg							7
Overall Accuracy (%)		**84**						

2016 area (m2)		Observed						
		BareSoil	HighVegetation	LowVegetation	IsolatedSmall	IsolatedSmallBareSoil	IsolatedSmallLowVeg	IsolatedLowVeg
Predicted	BareSoil	232.5						
	HighVegetation		1484.0	11.6				
	LowVegetation		69.2	643.1				
	IsolatedSmall						14.6	
	IsolatedSmallBareSoil					7.3		
	IsolatedSmallLowVeg			23.9			21.4	
	IsolatedLowVeg							124.6
Overall Accuracy (%)		**95.5**						

2016 count	Producer's Accuracy (%)	User's Accuracy (%)
BareSoil	100.0	100.0
HighVegetation	45.0	75.0
LowVegetation	89.2	75.0
IsolatedSmall	-	0.0
IsolatedSmallBareSoil	100.0	100.0
IsolatedSmallLowVeg	95.8	95.8
IsolatedLowVeg	100.0	100.0

2016 area (m2)	Producer's Accuracy (%)	User's Accuracy (%)
BareSoil	100.0	100.0
HighVegetation	95.5	99.2
LowVegetation	94.8	90.3
IsolatedSmall	-	0.0
IsolatedSmallBareSoil	100.0	100.0
IsolatedSmallLowVeg	59.5	47.3
IsolatedLowVeg	100.0	100.0

Appendix C4. Classification Accuracy Assessment 2019

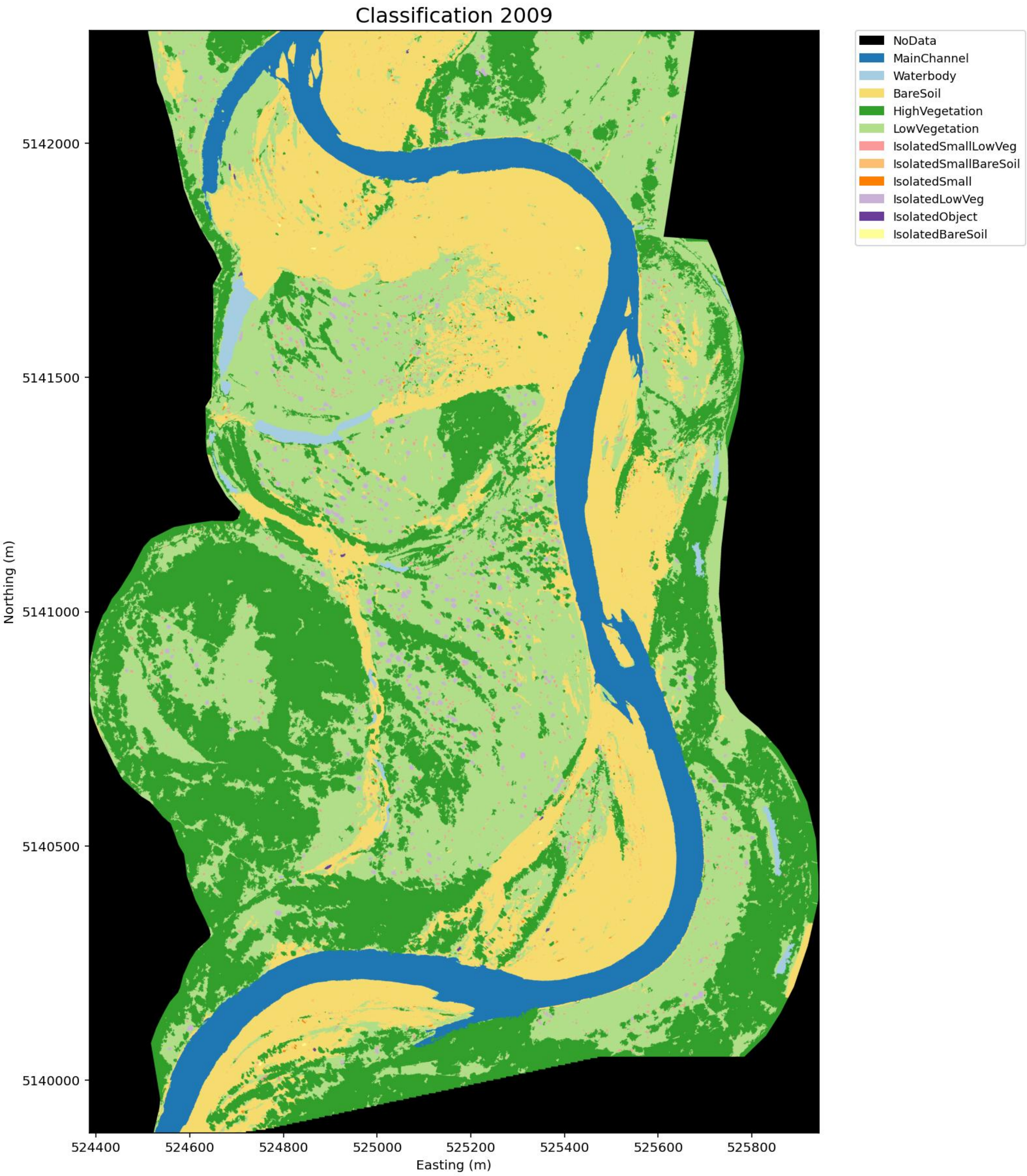
2019 count		Observed							
		BareSoil	HighVegetation	LowVegetation	IsolatedSmall	IsolatedSmallBareSoil	IsolatedSmallLowVeg	IsolatedLowVeg	IsolatedObject
Predicted	BareSoil	8							
	HighVegetation		10	1					
	LowVegetation	2	7	23					
	IsolatedSmall				5		1		
	IsolatedSmallBareSoil					5			
	IsolatedSmallLowVeg			1		1	24		
	IsolatedLowVeg							10	
	IsolatedObject								2
Overall Accuracy (%)		87							

2019 area (m2)		Observed							
		BareSoil	HighVegetation	LowVegetation	IsolatedSmall	IsolatedSmallBareSoil	IsolatedSmallLowVeg	IsolatedLowVeg	IsolatedObject
Predicted	BareSoil	605.1							
	HighVegetation		1564.5	200.1					
	LowVegetation	43.7	129.2	5018.2					
	IsolatedSmall				74.7		11.3		
	IsolatedSmallBareSoil					24.1			
	IsolatedSmallLowVeg			15.8		2.3	257.1		
	IsolatedLowVeg							554.1	
	IsolatedObject								97.6
Overall Accuracy (%)		95.3							

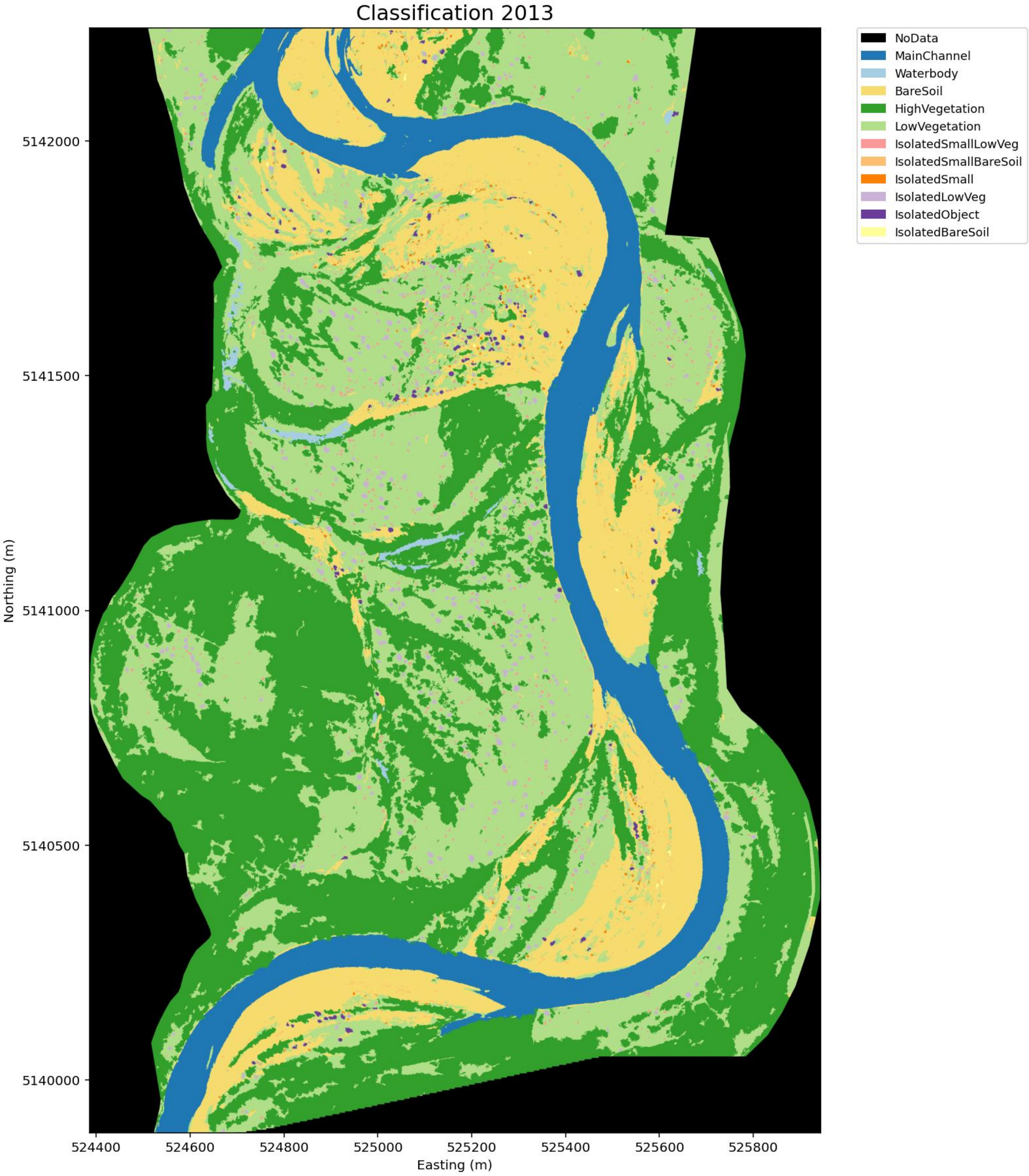
2019 count	Producer's Accuracy (%)	User's Accuracy (%)
BareSoil	80.0	100.0
HighVegetation	58.8	90.9
LowVegetation	92.0	71.9
IsolatedSmall	100.0	83.3
IsolatedSmallBareSoil	83.3	100.0
IsolatedSmallLowVeg	96.0	92.3
IsolatedLowVeg	100.0	100.0
IsolatedObject	100.0	100.0

2019 area (m2)	Producer's Accuracy (%)	User's Accuracy (%)
BareSoil	93.3	100.0
HighVegetation	92.4	88.7
LowVegetation	95.9	96.7
IsolatedSmall	100.0	86.9
IsolatedSmallBareSoil	91.2	100.0
IsolatedSmallLowVeg	95.8	93.4
IsolatedLowVeg	100.0	100.0
IsolatedObject	100.0	100.0

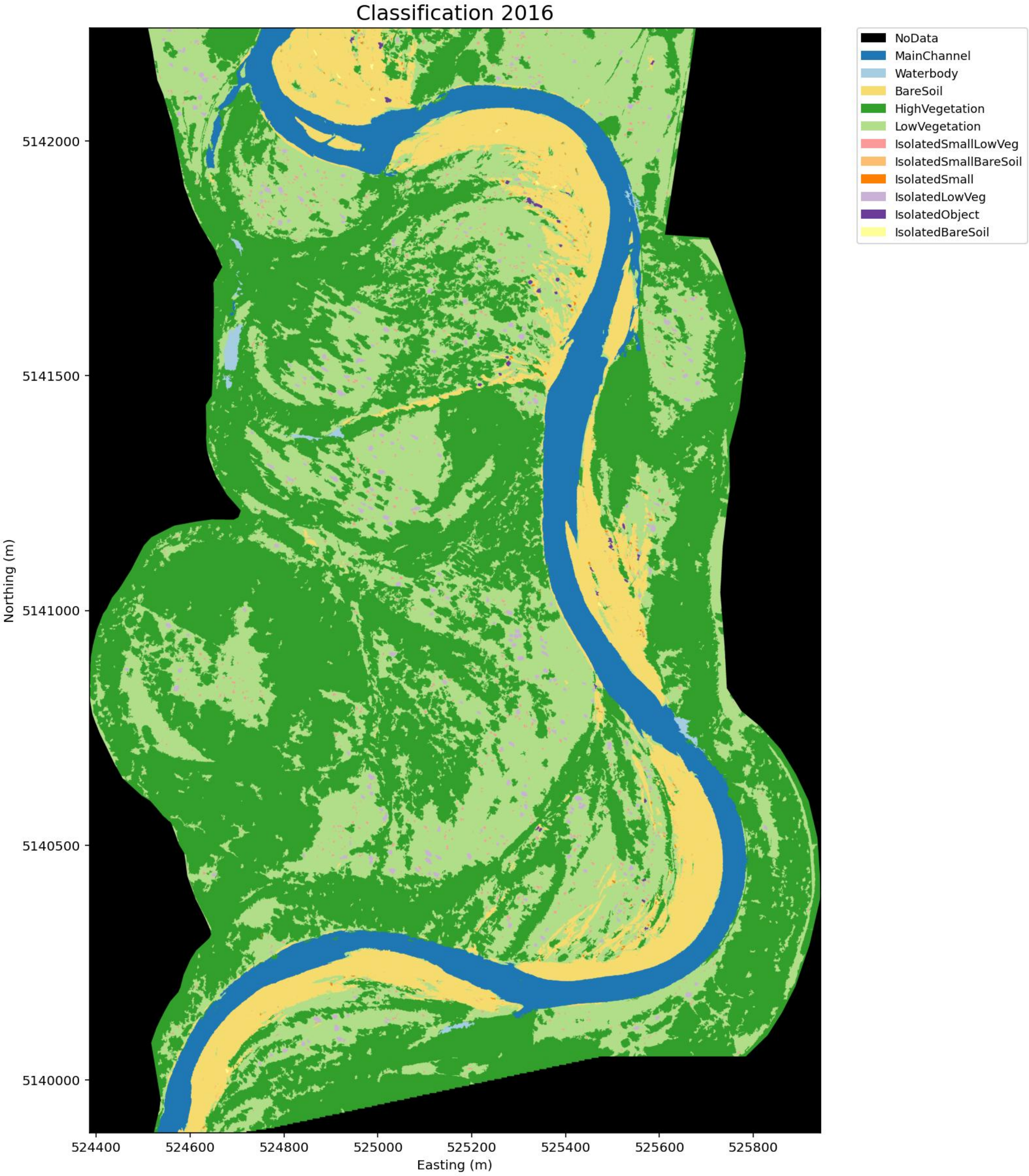
Appendix D1. Full size classification 2009



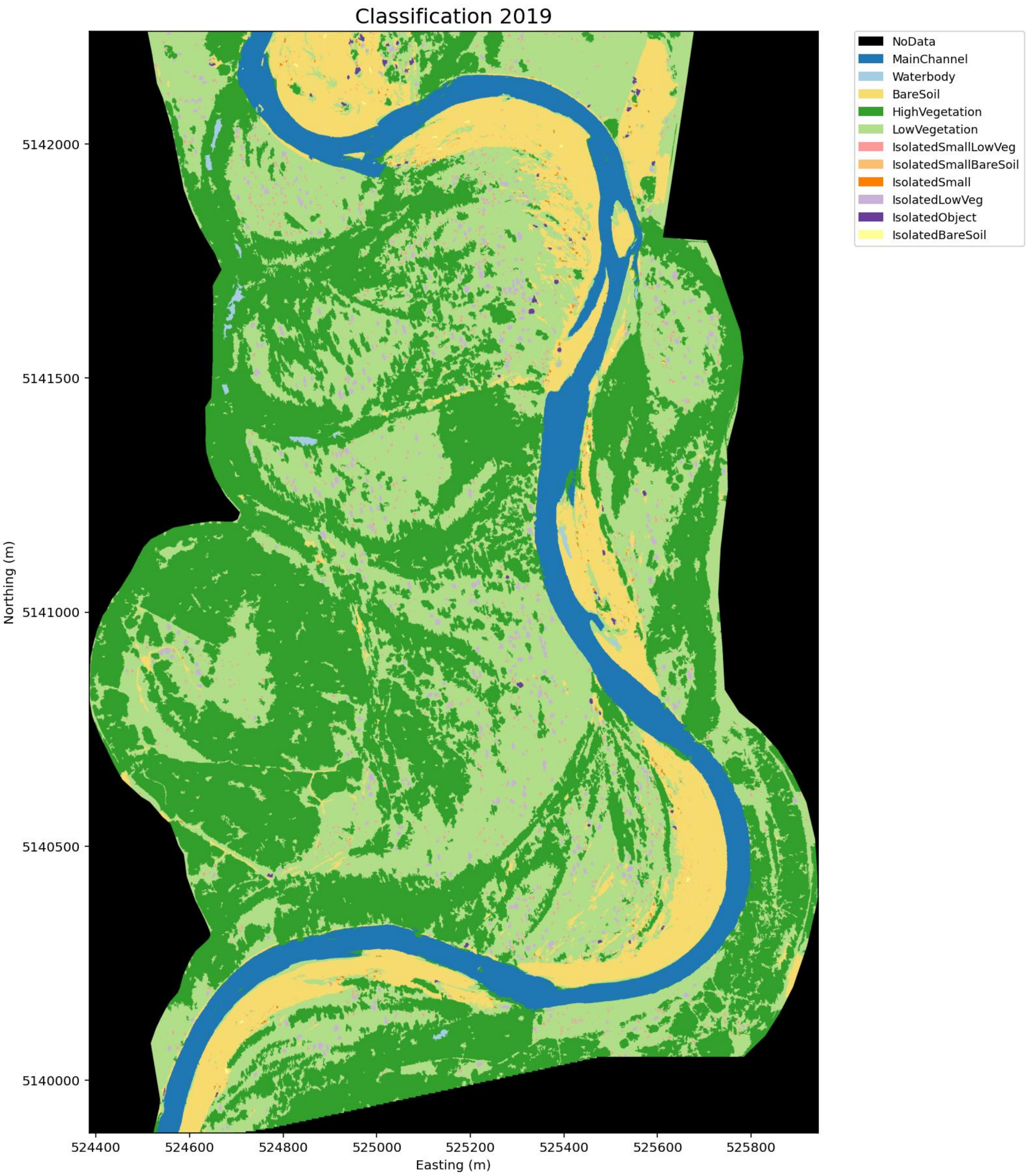
Appendix D2. Full size classification 2013



Appendix D3. Full size classification 2016



Appendix D4. Full size classification 2019



Appendix E. Python script Temporal Network

```
# Import packages
from os.path import join as pjoin
import pandas as pd
import numpy as np
import osgeo
import xarray as xr
import xrspatial as xrs
import rioxarray
import os
import rasterio
from rasterio.crs import CRS
from rasterio.transform import from_origin
from matplotlib.patches import Patch
import matplotlib.ticker as ticker
from shapely.geometry import Point
from collections import Counter, defaultdict
import math
import csv

os.environ['USE_PYGEOS'] = '0'
import geopandas as gpd
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap, BoundaryNorm, Normalize
import seaborn as sns
import networkx as nx

# directory settings
# Folders
raster_folder =
r"C:\Users\timme\OneDrive\Bureaublad\Master\Master_Thesis\rasterized_Full_classifications_subsets\subset3"
shapefile_folder = r"C:\Users\timme\OneDrive\Bureaublad\Master\Master_Thesis\Full_classifications"
output_folder =
r"C:\Users\timme\OneDrive\Bureaublad\Master\Master_Thesis\rasterized_Full_classifications_subsets\subset3_output"

# Define years
years = [1946, 1954, 1960, 1968, 1975, 1980, 1985, 1992, 1998, 2000, 2002, 2009, 2013, 2016, 2019]

#%%

#Import raster subset and join with shapefile attribute table
os.makedirs(output_folder, exist_ok=True)

# Loop over raster files
for raster_filename in os.listdir(raster_folder):
  if not raster_filename.endswith(".tif"):
    continue

  raster_path = os.path.join(raster_folder, raster_filename)

  # Infer corresponding shapefile name (assuming same base name but .shp)
  base_name = os.path.splitext(raster_filename)[0]

  output_csv = os.path.join(output_folder, base_name + "_merged.csv")

  if os.path.exists(output_csv):
    print(f"Skipping {raster_filename}: merged CSV already exists.")
    continue

  shapefile_name = base_name + ".shp"
  shapefile_path = os.path.join(shapefile_folder, shapefile_name)
```

```
    if not os.path.exists(shapefile_path):
        print(f"Skipping {raster_filename} — no matching shapefile found.")
        continue

    # Load shapefile and prepare FID
    gdf = gpd.read_file(shapefile_path)
    gdf["FID"] = gdf.index.astype("int32")

    if "Shape_inde" in gdf.columns:
        gdf.rename(columns={"Shape_inde": "Shape_index"}, inplace=True)
        gdf.to_file(shapefile_path)
        print(f"Renamed column in: {shapefile_name}")
    else:
        print(f"No change needed for: {shapefile_name}")

    # Load raster and with rioxarray get unique FIDs
    raster = rioxarray.open_rasterio(raster_path, masked=True).squeeze()

    #define coordinate system
    crs = CRS.from_string("EPSG:32631")
    raster = raster.rio.reproject(crs)

    # Set a valid integer nodata value
    raster.rio.write_nodata(0, inplace=True)

    # Convert raster to dataframe including coordinates and values
    raster_df = raster.to_dataframe(name="FID").reset_index()

    # Remove nodata or invalid values (assuming nodata=0 or mask applied)
    raster_df = raster_df[raster_df["FID"].notna()]
    raster_df["FID"] = raster_df["FID"].astype(int)

    # Join raster with attributes
    merged_df = raster_df.merge(gdf.drop(columns="geometry"), on="FID", how="left")

    # Class name to ID mapping
    class_mapping = {
        "MainChannel": 1,
        "Waterbody": 2,
        "BareSoil": 3,
        "HighVegetation": 4,
        "LowVegetation": 5,
        "IsolatedSmallLowVeg": 6,
        "IsolatedSmallBareSoil": 7,
        "IsolatedSmall": 8,
        "IsolatedLowVeg": 9,
        "Isolated Object": 10,
        "IsolatedBareSoil": 11
    }
    # Add Class_ID column based on Class_name
    merged_df["Class_ID"] = merged_df["Class_name"].map(class_mapping)

    print(f"Merged attributes for {raster_filename}:")
    print(merged_df.head())  # Print first few rows for brevity

    # Optionally save merged dataframe for further processing:
    output_csv = os.path.join(output_folder, base_name + "_merged.csv")
    merged_df.to_csv(output_csv, index=False)

#%%

# Function: Convert dataframe to raster
def dataframe_to_raster(merged_df, output_path, pixel_size=0.3, crs="EPSG:32631", nodata_value=0):
```

```
    x_unique = np.sort(merged_df['x'].unique())
    y_unique = np.sort(merged_df['y'].unique())[::-1]  # top to bottom

    x_to_col = {x: i for i, x in enumerate(x_unique)}
    y_to_row = {y: i for i, y in enumerate(y_unique)}

    raster_array = np.full((len(y_unique), len(x_unique)), nodata_value, dtype=np.uint8)

    for _, row in merged_df.iterrows():
        x, y, class_id = row['x'], row['y'], row['Class_ID']
        if pd.notna(class_id):
            row_idx = y_to_row[y]
            col_idx = x_to_col[x]
            raster_array[row_idx, col_idx] = int(class_id)

    transform = from_origin(x_unique.min() - pixel_size / 2, y_unique.max() + pixel_size / 2, pixel_size, pixel_size)

    with rasterio.open(
        output_path, "w",
        driver="GTiff",
        height=raster_array.shape[0],
        width=raster_array.shape[1],
        count=1,
        dtype=raster_array.dtype,
        crs=crs,
        transform=transform,
        nodata=nodata_value
    ) as dst:
        dst.write(raster_array, 1)

# Rasterize each year's merged dataframe
for year in years:
    csv_path = os.path.join(output_folder, f"Allier_{year}_FullClassification_merged.csv")
    out_raster = os.path.join(output_folder, f"classified_{year}.tif")

    if not os.path.exists(csv_path):
        print(f"Skipped {year}: CSV not found.")
        continue

    if os.path.exists(out_raster):
        print(f"Skipping rasterization for {year}: raster already exists.")
        continue

    df = pd.read_csv(csv_path)

    # Rasterize
    print(f"Rasterizing {year}...")
    dataframe_to_raster(df, out_raster)

print("Done.")

#%%

def create_class_cmap():
    """
    Create a class table, colormap, and normalization for plotting classified rasters.
    Returns:
        class_table (pd.DataFrame): Table with Class_ID, Class_name, and Color
        cmap (ListedColormap): Colormap object for plotting
        norm (BoundaryNorm): Normalization object for plotting
        labels (np.ndarray): Array of class names
    """
    class_info = [
```

```
        {"Class_ID": 0, "Class_name": "NoData", "Color": "#000000"},
        {"Class_ID": 1, "Class_name": "MainChannel", "Color": "#1f78b4"},
        {"Class_ID": 2, "Class_name": "Waterbody", "Color": "#a6cee3"},
        {"Class_ID": 3, "Class_name": "BareSoil", "Color": "#f7dc6f"},
        {"Class_ID": 4, "Class_name": "HighVegetation", "Color": "#33a02c"},
        {"Class_ID": 5, "Class_name": "LowVegetation", "Color": "#b2df8a"},
        {"Class_ID": 6, "Class_name": "IsolatedSmallLowVeg", "Color": "#fb9a99"},
        {"Class_ID": 7, "Class_name": "IsolatedSmallBareSoil", "Color": "#fdbf6f"},
        {"Class_ID": 8, "Class_name": "IsolatedSmall", "Color": "#ff7f00"},
        {"Class_ID": 9, "Class_name": "IsolatedLowVeg", "Color": "#cab2d6"},
        {"Class_ID": 10, "Class_name": "IsolatedObject", "Color": "#6a3d9a"},
        {"Class_ID": 11, "Class_name": "IsolatedBareSoil", "Color": "#ffff99"},
    ]

    class_table = pd.DataFrame(class_info).sort_values('Class_ID')

    values = class_table['Class_ID'].values
    colors = class_table['Color'].values
    labels = class_table['Class_name'].values

    cmap = ListedColormap(colors)
    boundaries = [v - 0.5 for v in values] + [values[-1] + 0.5]
    norm = BoundaryNorm(boundaries, len(values))

    return class_table, cmap, norm, labels, colors

class_table, cmap, norm, labels, colors = create_class_cmap()

#%%
# Plot maps

# File paths for all years
file_paths = {
    year: os.path.join(output_folder, f"classified_{year}.tif")
    for year in years
}

n_years = len(file_paths)
fig, axes = plt.subplots(3, 5, figsize=(24, 14), sharey=True)  # 3 rows x 5 cols
axes = axes.flatten()

for ax, (year, path) in zip(axes, file_paths.items()):
    with rasterio.open(path) as src:
        data = src.read(1)
        transform = src.transform
        width = src.width
        height = src.height

        x_coords = np.arange(width) * transform.a + transform.c
        y_coords = np.arange(height) * transform.e + transform.f

        im = ax.imshow(data, cmap=cmap, norm=norm,
                       extent=[x_coords[0], x_coords[-1], y_coords[-1], y_coords[0]])
        ax.set_title(f"Classification {year}")
        ax.set_xlabel("Easting (m)")
        ax.xaxis.set_major_formatter(ticker.FuncFormatter(lambda x, _: f"{int(x)}"))
        ax.yaxis.set_major_formatter(ticker.FuncFormatter(lambda y, _: f"{int(y)}"))

axes[0].set_ylabel("Northing (m)")

# Hide unused axes if any
for i in range(len(file_paths), len(axes)):
    axes[i].axis('off')
```

```
# Shared legend
legend_patches = [Patch(color=color, label=label) for color, label in zip(colors, labels)]
fig.legend(handles=legend_patches, bbox_to_anchor=(1.02, 0.5), loc='center left')

plt.tight_layout()
plt.subplots_adjust(right=0.88)
plt.show()

#%%
# extent
# Loop over rasters
for year, path in file_paths.items():
    if not os.path.exists(path):
        print(f"[!] Missing: {year} → {path}")
        continue

    raster = rioxarray.open_rasterio(path).squeeze()  # remove band dimension
    extent = raster.rio.bounds()
    shape = raster.shape  # (rows, cols)

    print(f"Year {year} → Extent: {extent}, Shape: {shape}")

#%%

#creating 3D stack of maps

# Pivot DataFrames to align spatial dimensions
def pivot_data(df, attribute):
    return df.pivot(index='y', columns='x', values=attribute).values

# Dictionary of merged DataFrames
merged_dfs = {
    year: pd.read_csv(os.path.join(output_folder, f"Allier_{year}_FullClassification_merged.csv"))
    for year in years
    if os.path.exists(os.path.join(output_folder, f"Allier_{year}_FullClassification_merged.csv"))
}

# Attributes you want to stack
attributes = ["FID", "Class_ID", "Area_in_m2", "Shape_index"]

# Dictionary to hold stacked arrays
stacked_arrays = {
    attr: np.stack([pivot_data(merged_dfs[year], attr) for year in years])
    for attr in attributes
}

#%%

#Create datset ds1

# Use the first available year to get spatial coordinates
first_year = next(iter(merged_dfs))
reference_df = merged_dfs[first_year]

# Extract sorted x and y coordinates from the reference DataFrame
x_coords = np.sort(reference_df['x'].unique())
y_coords = np.sort(reference_df['y'].unique())[::-1]  # reverse y for top-down order

# Create xarray Dataset
ds1 = xr.Dataset(
    data_vars={
        varname: (["year", "y", "x"], array)
```

```
        for varname, array in stacked_arrays.items()
    },
    coords={
        "x": x_coords,
        "y": y_coords,
        "year": years
    }
)

# Add CRS information
ds1.rio.write_crs("EPSG:32631", inplace=True)

# Print the Dataset summary
print(ds1)

#%%

# Export to geodataframe
df = ds1.to_dataframe().reset_index().drop('spatial_ref', axis=1)
df

#%%

#Create nodes

# Node is defined by the year and the clump_ID
# Determine the node size
nodes = df.groupby(['year', 'FID']).count()
nodes['ncells'] = nodes.x

# add the coordinates of the clump centroid to the nodes dataframe
centroids = df.groupby(['year', 'FID'])[["x", "y"]].mean()

# Merge original attributes from df
attributes = df.groupby(["year", "FID"])[["Class_ID", "Area_in_m2", "Shape_index"]].first()

nodes = nodes.merge(centroids, on=["year", "FID"])  # Merge centroids
nodes = nodes.merge(attributes, on=["year", "FID"])  # Merge original attributes

# Select the correct columns and rename them
nodes = nodes.rename(columns={
    "Class_ID_y": "Class_ID",
    "Area_in_m2_y": "Area_in_m2",
    "Shape_index_y": "Shape_index",
    "x_y": "xcoor",
    "y_y": "ycoor"
})
nodes = nodes.drop(columns=["y_x", "x_x", "Class_ID_x", "Area_in_m2_x", "Shape_index_x"])

nodes = nodes.reset_index()
#nodes

#%%

#Plot nodes
gdf = gpd.GeoDataFrame(data=nodes, geometry=gpd.points_from_xy(nodes.xcoor, nodes.ycoor), crs=ds1.rio.crs)
#gdf.plot(column = 'year', legend=True)

#%%

# Directional graph G
G = nx.DiGraph()
```

```
id_to_name = dict(zip(class_table["Class_ID"], class_table["Class_name"]))

for r in nodes.values:
    year, FID, ncells, xcoor, ycoor, Class_ID, Area_in_m2, Shape_index = r
    class_name = id_to_name.get(Class_ID, "Unknown")

    attributes = {
        "size": int(ncells),
        "xcoor": xcoor,
        "ycoor": ycoor,
        "year": int(year),
        "Class_ID": int(Class_ID),
        "Class_name": str(class_name),
        "Area_in_m2": float(Area_in_m2),
        "Shape_index": float(Shape_index)
    }

    if FID == 0:
        attributes["xcoor"] = 570000
        attributes["ycoor"] = 5800000

    G.add_node((int(year), int(FID)), **attributes)

#%%

# plot the Graph with default positions
veg_size = 0.01 * np.array(list(nx.get_node_attributes(G, 'size').values()))
#plt.figure()
#nx.draw(G, with_labels=True, node_size = veg_size)

#%%

# 2D plot of G
nodes_per_year = defaultdict(list)
for n in G.nodes:
    year = n[0]
    nodes_per_year[year].append(n)

# 2D plot
pos2D = {}
for year in sorted(nodes_per_year):
    # Sorteer objecten voor een consistente volgorde
    sorted_nodes = sorted(nodes_per_year[year], key=lambda x: x[1])  # sorteer op object ID
    for i, node in enumerate(sorted_nodes):
        # Zet op vaste verticale afstand
        pos2D[node] = np.array([year, i])

nx.draw(G, pos=pos2D, with_labels=True, node_size=veg_size)

#%%

# Determine the edges
gdf = ds1.to_dataframe().drop('spatial_ref', axis=1)
veg_time = gdf.pivot_table(values='FID', columns=['year'], index=['x', 'y'])
veg_time

#%%

edges_by_year_pair = {}

for i in range(len(years) - 1):
    year_start = years[i]
    year_end = years[i + 1]
```

```
    # Create edge transitions
    edges = veg_time[[year_start, year_end]].reset_index().fillna(0)
    edges = edges.groupby([year_start, year_end]).size().reset_index(name="count")

    # Store in a dictionary with real year keys
    edges_by_year_pair[(year_start, year_end)] = edges

    # Dynamically assign to a variable
    globals()[f"edges_{year_start}_{year_end}"] = edges

    print(f"Created edges_{year_start}_{year_end}")

#%%

class_color_map = dict(zip(class_table["Class_ID"], class_table["Color"]))

# Color per node
node_colors = []
for node in G.nodes():
    class_id = G.nodes[node].get("Class_ID", 0)
    color = class_color_map.get(class_id, "#000000")
    node_colors.append(color)

# prepare node labels
node_labels = {n: str(n) for n in G.nodes}
label_pos = {n: (x, y - 0.3) for n, (x, y) in pos2D.items()}

# prepare edge labels
edge_labels = {
    (u, v): d.get("weight", d.get("area_in_m2", ""))
    for u, v, d in G.edges(data=True)
}

# Plot
plt.figure(figsize=(12, 6))
nx.draw(
    G,
    pos=pos2D,
    with_labels=False,
    node_size=veg_size,
    node_color=node_colors,
    edge_color="gray",
    alpha=0.8,
    arrows=True
)

# Node labels
nx.draw_networkx_labels(G, pos=label_pos, labels=node_labels, font_size=11)

# Edge labels
nx.draw_networkx_edge_labels(G, pos=pos2D, edge_labels=edge_labels, font_size=7, label_pos=0.4)

plt.axis("off")
plt.tight_layout()
plt.show()

#%%

# plot in 3D with time
pos3D = {}
for n in G.nodes(data=True):
    node = n[0]  # (year, ID) tuple
```

```
    year = n[1]['year']
    x = n[1]['xcoor']
    y = n[1]['ycoor']
    pos3D[node] = (x, y, year)

#%%
if False:
    print(df.columns)

    def _format_axes(ax):
        """Visualization options for the 3D axes."""
        print("Formatting axes...")
        ax.grid(True)
        ax.set_xlim(df['x'].min(), df['x'].max())
        ax.set_ylim(df['y'].min(), df['y'].max())
        ax.set_zlim(df['year'].min() - 1, df['year'].max() + 1)
        print("Axes formatted.")

    def imshow3d(ax, array, xmin, ymin, cell_size, pos=0, norm=None, cmap=None, alpha=1.0):
        print(f"Plotting year at z={pos}...")
        if norm is None:
            norm = Normalize()
        colors = plt.get_cmap(cmap)(norm(array))
        if alpha < 1.0:
            colors = colors.copy()
            colors[..., 3] = alpha
        ny, nx = array.shape
        yi, xi = np.mgrid[0:ny + 1, 0:nx + 1]
        ymax = ymin + (ny * cell_size)
        yi = ymax - (yi * cell_size - cell_size / 2)
        xi = xi * cell_size + xmin - cell_size / 2
        zi = np.full_like(xi, pos)
        ax.plot_surface(xi, yi, zi, rstride=1, cstride=1,
                        facecolors=colors, shade=False, alpha=alpha)
        ax.view_init(10, 300)
        print(f"Finished plotting layer for year {pos}.")

    print("Extracting node and edge positions...")
    node_xyz = np.array([pos3D[v] for v in sorted(G)])
    edge_xyz = np.array([(pos3D[u], pos3D[v]) for u, v in G.edges()])
    print(f"Extracted {len(node_xyz)} nodes and {len(edge_xyz)} edges.")

    print("Creating figure and axes...")
    fig = plt.figure(figsize=(12,10))
    ax = fig.add_subplot(111, projection="3d")

    print("Calculating plot boundaries...")
    xmin = float(df['x'].min())
    ymin = float(df['y'].min())
    xmax = float(df['x'].max())
    ymax = float(df['y'].max())

    cell_size = float(df['x'][1] - df['x'][0])
    print(f"xmin={xmin}, xmax={xmax}, ymin={ymin}, ymax={ymax}, cell_size={cell_size}")

    print("Starting 3D surface plotting for each year...")
    for i, year in enumerate(years):
        print(f"Processing year {year} ({i+1}/{len(years)})...")
        class_data = ds1["Class_ID"].sel(year=year).values
        imshow3d(
            ax,
            class_data,
            xmin,
```

```
        ymin,
        cell_size,
        pos=year,
        norm=norm,
        cmap=cmap,
        alpha=0.6
    )

print("Finished plotting all years.")
print("Plotting nodes...")
ax.scatter(*node_xyz.T, s=150, color='red', ec='white', zorder=10)
print("Nodes plotted.")

print("Plotting edges...")
for vizedge in edge_xyz:
    ax.plot(*vizedge.T, color='cyan', linewidth=2, zorder=5)
print("Finished plotting edges.")

_format_axes(ax)
print("Plot complete.")
```

Appendix F. Python script transition types

```
# Import packages
from os.path import join as pjoin
import pandas as pd
import numpy as np
import osgeo
import xarray as xr
import xrspatial as xrs
import rioxarray
import os
import rasterio
from rasterio.crs import CRS
from rasterio.transform import from_origin
from matplotlib.patches import Patch
import matplotlib.ticker as ticker
from shapely.geometry import Point
from collections import Counter, defaultdict
import math
import csv
import pickle

os.environ['USE_PYGEOS'] = '0'
import geopandas as gpd
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap, BoundaryNorm, Normalize
import matplotlib.colors as mcolors
import seaborn as sns
import networkx as nx

# directory settings
# Folders
raster_folder =
r"C:\Users\timme\OneDrive\Bureaublad\Master\Master_Thesis\rasterized_Full_classifications_subsets\subset3"
shapefile_folder = r"C:\Users\timme\OneDrive\Bureaublad\Master\Master_Thesis\Full_classifications"
output_folder =
r"C:\Users\timme\OneDrive\Bureaublad\Master\Master_Thesis\rasterized_Full_classifications_subsets\subset3_output"

# Define years
years = [1946, 1954, 1960, 1968, 1975, 1980, 1985, 1992, 1998, 2000, 2002, 2009, 2013, 2016, 2019]
#%%
```

```
with open(os.path.join(output_folder, "graph_Gsubset3.gpickle"), "rb") as f:
    G = pickle.load(f)
#%%
def detect_growth_transitions(G):
    records = []

    for node in G.nodes:
        current = G.nodes[node]
        current_class = current.get("Class_ID")
        current_area = current.get("Area_in_m2", 0)
        current_year = current.get("year")
        current_HV = (current_class == 4)

        preds = list(G.predecessors(node))
        n_preds = len(preds)

        for pred in preds:
            parent = G.nodes[pred]
            parent_class = parent.get("Class_ID")
            parent_area = parent.get("Area_in_m2", 0)
            parent_year = parent.get("year")
            parent_HV = (parent_class == 4)

            if parent_HV and current_HV:
                if current_area > parent_area:
                    transition_direction = "growth"
                else:
                    transition_direction = "stable"
            elif not parent_HV and current_HV:
                transition_direction = "growth"
            else:
                transition_direction = "no transition"

            area_diff = current_area - parent_area

            records.append({
                "Node_t": node,
                "Node_tmin1": pred,
                "Year_t": current_year,
                "Year_tmin1": parent_year,
                "Class_t": current_class,
                "Class_tmin1": parent_class,
                "Area_t": current_area,
                "Area_tmin1": parent_area,
                "Area_diff": area_diff,
                "Transition": transition_direction,
                "n_nodes_tmin1": n_preds,
            })

    return pd.DataFrame(records)

def detect_degradation_transitions(G):
    records = []

    for node in G.nodes:
        current = G.nodes[node]
        current_class = current.get("Class_ID")
        current_area = current.get("Area_in_m2", 0)
        current_year = current.get("year")
        current_HV = (current_class == 4)

        succs = list(G.successors(node))
        n_succs = len(succs)
```

```
        for succ in succs:
            child = G.nodes[succ]
            child_class = child.get("Class_ID")
            child_area = child.get("Area_in_m2", 0)
            child_year = child.get("year")
            child_HV = (child_class == 4)

            if current_HV and child_HV:
                if child_area < current_area:
                    transition_direction = "degradation"
                else:
                    transition_direction = "stable"
            elif current_HV and not child_HV:
                transition_direction = "degradation"
            else:
                transition_direction = "no transition"

            area_diff = child_area - current_area

            records.append({
                "Node_t": node,
                "Node_tplus1": succ,
                "Year_t": current_year,
                "Year_tplus1": child_year,
                "Class_t": current_class,
                "Class_tplus1": child_class,
                "Area_t": current_area,
                "Area_tplus1": child_area,
                "Area_diff": area_diff,
                "Transition": transition_direction,
                "n_nodes_tplus1": n_succs,
            })

    return pd.DataFrame(records)

#%%
# Run transition detection functions
df_growth = detect_growth_transitions(G)
df_degradation = detect_degradation_transitions(G)

# Add missing columns for consistency
df_growth["n_nodes_tmin1"] = df_growth["n_nodes_tmin1"].fillna(0).astype(int)
df_growth["n_nodes_tplus1"] = 0  # growth doesn't look ahead

df_degradation["n_nodes_tmin1"] = 0  # degradation doesn't look back
df_degradation["n_nodes_tplus1"] = df_degradation["n_nodes_tplus1"].fillna(0).astype(int)

# Combine both transition DataFrames
df_alltransitions = pd.concat([df_growth, df_degradation], ignore_index=True)

# Keep only relevant transitions with HighVegetation involvement
df_HighVeg = df_alltransitions[
    (df_alltransitions["Class_t"] == 4) |
    (df_alltransitions["Class_tmin1"] == 4) |
    (df_alltransitions["Class_tplus1"] == 4)
].copy()
#%%
# Remap to grouped class system: 1 = HV, 2 = nonHV, 3 = isoHV
class_remap = {
    1: 2,  # MainChannel → nonHV
    2: 2,  # Waterbody → nonHV
    3: 2,  # BareSoil → nonHV
```

```
    4: 1,  # HighVegetation → HV
    5: 2,  # LowVegetation → nonHV
    6: 3,  # IsolatedSmallLowVeg → isoHV
    7: 3,  # IsolatedSmallBareSoil → isoHV
    8: 3,  # IsolatedSmall → isoHV
    9: 3,  # IsolatedLowVeg → isoHV
    10: 3, # IsolatedObject → isoHV
    11: 3  # IsolatedBareSoil → isoHV
}

df_HighVeg["Class_t_new"]     = df_HighVeg["Class_t"].map(class_remap)
df_HighVeg["Class_tmin1_new"]   = df_HighVeg["Class_tmin1"].map(class_remap)
df_HighVeg["Class_tplus1_new"]   = df_HighVeg["Class_tplus1"].map(class_remap)

# Classify transitions
growth_direction = (
    (df_HighVeg["Transition"] == "growth") &
    (df_HighVeg["Class_t_new"] == 1)  # end state = HV
)

degradation_direction = (
    (df_HighVeg["Transition"] == "degradation") &
    (df_HighVeg["Class_t_new"] == 1)  # start state = HV
)

stable_direction = (
    (df_HighVeg["Transition"] == "stable") &
    (df_HighVeg["Class_t_new"] == 1) &
    ((df_HighVeg["Class_tmin1_new"] == 1) | (df_HighVeg["Class_tplus1_new"] == 1))
)

growth = df_HighVeg[growth_direction].copy()
degradation = df_HighVeg[degradation_direction].copy()
stable = df_HighVeg[stable_direction].copy()

print("Growth examples:\n", growth.head())
print("Degradation examples:\n", degradation.head())
print("Stable examples:\n", stable.head())


#%%
###GROWTH###

# Initialize
growth["growth_type"] = "none"

# Group by Node to handle multiple parents
growth_grouped = growth.groupby("Node_t").agg({
    "Node_tmin1": lambda x: list(x),
    "Class_tmin1_new": lambda x: list(x),
    "Class_t_new": "first",  # All same in filtered data
    "growth_type": "first"
}).reset_index()

# Add current area
growth_grouped = growth_grouped.merge(
    growth[["Node_t", "Area_t"]].drop_duplicates("Node_t"),
    on="Node_t",
    how="left"
)

# Prepare parent area data
parent_areas = growth[["Node_t", "Node_tmin1", "Area_tmin1"]].explode("Node_tmin1")
```

```
parent_areas_grouped = parent_areas.groupby("Node_t")["Area_tmin1"].sum().reset_index()

# Merge total parent area into growth_grouped
growth_grouped = growth_grouped.merge(parent_areas_grouped, on="Node_t", how="left")

# Count parent types per group
growth_grouped["n_HV"] = growth_grouped["Class_tmin1_new"].apply(lambda x: x.count(1))
growth_grouped["n_isoHV"] = growth_grouped["Class_tmin1_new"].apply(lambda x: x.count(3))
growth_grouped["n_nonHV"] = growth_grouped["Class_tmin1_new"].apply(lambda x: x.count(2))

# Define conditions
growth_conditions = [
    #Expansion
    (growth_grouped["Class_t_new"] == 1) & (growth_grouped["n_HV"] == 1)
    & (growth_grouped["n_isoHV"] == 0) & (growth_grouped["n_nonHV"] >= 0),
    #Eating
    (growth_grouped["Class_t_new"] == 1) & (growth_grouped["n_HV"] == 1)
    & (growth_grouped["n_isoHV"] >= 1) & (growth_grouped["n_nonHV"] >= 0),
    #Merging
    (growth_grouped["Class_t_new"] == 1) & (growth_grouped["n_HV"] > 1)
    & (growth_grouped["n_isoHV"] == 0) & (growth_grouped["n_nonHV"] >= 0),
    #Eating & Merging
    (growth_grouped["Class_t_new"] == 1) & (growth_grouped["n_HV"] > 1)
    & (growth_grouped["n_isoHV"] >= 1) & (growth_grouped["n_nonHV"] >= 0),
    #Emerging from one isoHV
    (growth_grouped["Class_t_new"] == 1) & (growth_grouped["n_HV"] == 0)
    & (growth_grouped["n_isoHV"] == 1) & (growth_grouped["n_nonHV"] >= 0),
    #Emerging from multiple isoHV
    (growth_grouped["Class_t_new"] == 1) & (growth_grouped["n_HV"] == 0)
    & (growth_grouped["n_isoHV"] > 1) & (growth_grouped["n_nonHV"] >= 0),
    #Emerging from nonHV
    (growth_grouped["Class_t_new"] == 1) & (growth_grouped["n_HV"] == 0)
    & (growth_grouped["n_isoHV"] == 0) & (growth_grouped["n_nonHV"] >= 1),
]

growth_choices = [
    'Expansion',
    'Eating',
    'Merging',
    'Eating & Merging',
    'Emerging from one isoHV',
    'Emerging from multiple isoHV',
    'Emerging from nonHV'
    ]

# Apply growth type classification
growth_grouped["growth_type"] = np.select(growth_conditions, growth_choices, default="none")

# Extract year from Node
growth_grouped["Node_t"] = growth_grouped["Node_t"].apply(lambda t: (int(t[0]), int(t[1])))
growth_grouped["year"] = growth_grouped["Node_t"].apply(lambda t: t[0])

# Print summary
print("Growth Table:")
print(growth_grouped[['Node_t', 'Node_tmin1', 'Class_tmin1_new', 'Class_t_new', 'growth_type']])

growcount = growth_grouped.groupby("growth_type")["Class_tmin1_new"].count().rename("count")
print("\nGrowth Type Counts:")
print(growcount)
# Plot Growth Transition Types
def plot_growth_types(growcount):
    growcount_sorted = growcount.sort_values(ascending=False)
    plt.figure(figsize=(8, 5))
```

```
    plt.bar(growcount_sorted.index, growcount_sorted.values, color='forestgreen')
    plt.title('Count of Growth Transition Types')
    plt.ylabel('Count')
    plt.xlabel('Growth Type')
    plt.xticks(rotation=45, ha='right')
    plt.grid(axis='y', linestyle='--', alpha=0.7)
    plt.tight_layout()
    plt.show()

plot_growth_types(growcount)

# Save to CSV
growth_table_path = os.path.join(output_folder, "Grouped_Growth_Table.csv")
growth_grouped.to_csv(growth_table_path, index=False, quoting=csv.QUOTE_ALL)

#Add growth variables to G
# Loop through each row in the DataFrame
for row in growth_grouped.itertuples(index=False):
    node_key = row.Node_t
    if node_key in G.nodes:
        G.nodes[node_key].update({
            "Node_tmin1": row.Node_tmin1,
            "Class_tmin1_new": row.Class_tmin1_new,
            "Class_t_new": row.Class_t_new,
            "growth_type": row.growth_type,
            "Area_t": row.Area_t,
            "Area_tmin1": row.Area_tmin1,
            "n_HV": row.n_HV,
            "n_isoHV": row.n_isoHV,
            "n_nonHV": row.n_nonHV
        })

#Find all growth nodes
growth_nodes = [n for n, d in G.nodes(data=True) if 'growth_type' in d]

#%%
###DEGRADATION###

degradation["degradation_type"] = "none"

# Group by Node (i.e., parent) to handle multiple children
degradation_grouped = degradation.groupby("Node_t").agg({
    "Node_tplus1": lambda x: list(x),
    "Class_tplus1_new": lambda x: list(x),
    "Class_t_new": "first",  # all same in filtered data
    "degradation_type": "first"
}).reset_index()

# Add Area_t from one of the parents (all the same in grouped data)
degradation_grouped = degradation_grouped.merge(
    degradation[["Node_t", "Area_t"]].drop_duplicates("Node_t"),
    on="Node_t",
    how="left"
)

# Prepare child area data
child_areas = degradation[["Node_t", "Node_tplus1", "Area_tplus1"]].explode("Node_tplus1")
child_areas_grouped = child_areas.groupby("Node_t")["Area_tplus1"].sum().reset_index()

# Merge total child area into degradation_grouped
degradation_grouped = degradation_grouped.merge(child_areas_grouped, on="Node_t", how="left")

# Count child class types
```

```
degradation_grouped["n_HV"] = degradation_grouped["Class_tplus1_new"].apply(lambda x: x.count(1))
degradation_grouped["n_isoHV"] = degradation_grouped["Class_tplus1_new"].apply(lambda x: x.count(3))
degradation_grouped["n_nonHV"] = degradation_grouped["Class_tplus1_new"].apply(lambda x: x.count(2))


# Define conditions for degradation
degradation_conditions = [
    #Shrinking
    (degradation_grouped["Class_t_new"] == 1) & (degradation_grouped["n_HV"] == 1)
    & (degradation_grouped["n_isoHV"] == 0) & (degradation_grouped["n_nonHV"] >= 0),
    #Shattering
    (degradation_grouped["Class_t_new"] == 1) & (degradation_grouped["n_HV"] == 1)
    & (degradation_grouped["n_isoHV"] >= 1) & (degradation_grouped["n_nonHV"] >= 0),
    #Splitting
    (degradation_grouped["Class_t_new"] == 1) & (degradation_grouped["n_HV"] > 1)
    & (degradation_grouped["n_isoHV"] == 0) & (degradation_grouped["n_nonHV"] >= 0),
    #Shattering & Splitting
    (degradation_grouped["Class_t_new"] == 1) & (degradation_grouped["n_HV"] > 1)
    & (degradation_grouped["n_isoHV"] >= 1) & (degradation_grouped["n_nonHV"] >= 0),
    #Shrink to one isoHV
    (degradation_grouped["Class_t_new"] == 1) & (degradation_grouped["n_HV"] == 0)
    & (degradation_grouped["n_isoHV"] == 1) & (degradation_grouped["n_nonHV"] >= 0),
    #Shrink to multiple isoHV
    (degradation_grouped["Class_t_new"] == 1) & (degradation_grouped["n_HV"] == 0)
    & (degradation_grouped["n_isoHV"] > 1) & (degradation_grouped["n_nonHV"] >= 0),
    #Shrink to nonHV
    (degradation_grouped["Class_t_new"] == 1) & (degradation_grouped["n_HV"] == 0)
    & (degradation_grouped["n_isoHV"] == 0) & (degradation_grouped["n_nonHV"] >= 1),
]

degradation_choices = [
    'Shrinking',
    'Shattering',
    'Splitting',
    'Shattering & Splitting',
    'Shrink to one isoHV',
    'Shrink to multiple isoHV',
    'Shrink to nonHV'
    ]

# Assign degradation type
degradation_grouped["degradation_type"] = np.select(degradation_conditions, degradation_choices, default="none")
# Format children
degradation_grouped["Node_t"] = degradation_grouped["Node_t"].apply(lambda t: (int(t[0]), int(t[1])))
degradation_grouped["year"] = degradation_grouped["Node_t"].apply(lambda t: t[0])


# Add year
degradation_grouped["year"] = degradation_grouped["Node_t"].apply(lambda t: t[0])

# Print summary
print("\nDegradation Table:")
print(degradation_grouped[['Node_t', 'Node_tplus1', 'Class_t_new', 'Class_tplus1_new', 'degradation_type']])

# Print type counts
degradationcount = degradation_grouped.groupby('degradation_type')['Class_t_new'].count().rename('count')
print("\nDegradation Type Counts:")
print(degradationcount)

def plot_degradation_types(degradationcount):
    degradationcount_sorted = degradationcount.sort_values(ascending=False)
    plt.figure(figsize=(8, 5))
    plt.bar(degradationcount_sorted.index, degradationcount_sorted.values, color='firebrick')
```

```
    plt.title('Count of Degradation Transition Types')
    plt.ylabel('Count')
    plt.xlabel('Degradation Type')
    plt.xticks(rotation=45, ha='right')
    plt.grid(axis='y', linestyle='--', alpha=0.7)
    plt.tight_layout()
    plt.show()

plot_degradation_types(degradationcount)

# Save to CSV
degradation_table_path = os.path.join(output_folder, "Grouped_Degradation_Table.csv")
degradation_grouped.to_csv(degradation_table_path, index=False, quoting=csv.QUOTE_ALL)

# Add degradation attributes to network G
for row in degradation_grouped.itertuples(index=False):
    node_key = row.Node_t
    if node_key in G.nodes:
        G.nodes[node_key].update({
            'Node_tplus1': row.Node_tplus1,
            'Class_tplus1_new': row.Class_tplus1_new,
            'Class_t_new': row.Class_t_new,
            'degradation_type': row.degradation_type,
            'Area_t': row.Area_t,
            'Area_tplus1': row.Area_tplus1,
            'n_HV': row.n_HV,
            'n_isoHV': row.n_isoHV,
            'n_nonHV': row.n_nonHV
        })

# Find all degradation nodes
degradation_nodes = [n for n, d in G.nodes(data=True) if 'degradation_type' in d]
#%%
import matplotlib.pyplot as plt

def plot_transition_types_side_by_side(growcount, degradationcount):
    # Sort for visual clarity
    growcount_sorted = growcount.sort_values(ascending=False)
    degradationcount_sorted = degradationcount.sort_values(ascending=False)

    # Set up subplots
    fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(14, 5), sharey=True)

    # Growth bar chart
    ax1.bar(growcount_sorted.index, growcount_sorted.values, color='forestgreen')
    ax1.set_title('Growth Transition Types')
    ax1.set_ylabel('Count')
    ax1.set_xlabel('Growth Type')
    ax1.set_xticklabels(growcount_sorted.index, rotation=45, ha='right')
    ax1.grid(axis='y', linestyle='--', alpha=0.7)

    # Degradation bar chart
    ax2.bar(degradationcount_sorted.index, degradationcount_sorted.values, color='firebrick')
    ax2.set_title('Degradation Transition Types')
    ax2.set_xlabel('Degradation Type')
    ax2.set_xticklabels(degradationcount_sorted.index, rotation=45, ha='right')
    ax2.grid(axis='y', linestyle='--', alpha=0.7)

    # Final layout
    plt.tight_layout()
    plt.show()

# Call the function
```

```
plot_transition_types_side_by_side(growcount, degradationcount)

#%%
###STABLE###
# Assume stable DataFrame columns: 'Node_t', 'Node_t+1', 'Class_t', 'Class_t+1', 'Area_t', 'Area_t+1', etc.

stable["stable_type"] = "Stable"

# Count stable nodes by stable_type
stablecount = stable.groupby("stable_type")["Class_t_new"].count().rename("count")

stable_table_path = os.path.join(output_folder, "Grouped_Stable_Table.csv")
print("\nStable Table:")
print(stable[['Node_t', "Node_tplus1", 'Class_t_new', 'Class_tplus1_new', 'stable_type']])

stable.to_csv(stable_table_path, index=False, quoting=csv.QUOTE_ALL)

print("\nStable Type Counts:")
print(stablecount)

# Add stable variables to graph G
for row in stable.itertuples(index=True):
    node_key = row.Node_t  # current node

    if node_key in G.nodes:
        G.nodes[node_key].update({
            'Node_tplus1': row.Node_tplus1,
            'Class_tplus1_new': row.Class_tplus1_new,
            'Class_t_new': row.Class_t_new,
            'stable_type': row.stable_type,
            'Area_tplus1': getattr(row, 'Area_tplus1', None),
            'Area_t': getattr(row, 'Area_t', None),
            'n_HV': getattr(row, 'n_parents', None),  # if applicable
            'n_isoHV': 0,  # adjust if needed
            'n_nonHV': 0
        })

# Find all stable nodes
stable_nodes = [n for n, d in G.nodes(data=True) if d.get('stable_type') == 'Stable']

#%%
#All nodes
df_nodes = pd.DataFrame.from_dict(dict(G.nodes(data=True)), orient='index')
df_nodes[['year', 'ID']] = pd.DataFrame(df_nodes.index.to_list(), index=df_nodes.index)
cols = ['year', 'ID'] + [col for col in df_nodes.columns if col not in ['year', 'ID']]
df_nodes = df_nodes[cols]

nodes_table_path = os.path.join(output_folder, "Nodes_Table.csv")
df_nodes.to_csv(nodes_table_path, index=False, quoting=csv.QUOTE_ALL)

#Only nodes with transitions
df_transitions = pd.DataFrame.from_dict(
    {k: v for k, v in G.nodes(data=True) if any(key in v for key in ["growth_type", "degradation_type", "stable_type"])},
    orient='index'
)
df_transitions[['year', 'ID']] = pd.DataFrame(df_transitions.index.to_list(), index=df_transitions.index)
cols = ['year', 'ID'] + [col for col in df_transitions.columns if col not in ['year', 'ID']]
df_transitions = df_transitions[cols]

transitions_table_path = os.path.join(output_folder, "Transitions_Table.csv")
df_transitions.to_csv(transitions_table_path, index=False, quoting=csv.QUOTE_ALL)
############################################
#%%
```

Appendix G. Python script lifecycles

```
#%%
#Object history choice for end object
def get_all_histories(graph, central_node):
    """Search for all paths from central_node to previous objects (multiple parents possible)"""
    histories = []

    def dfs(path):
        current = path[-1]
        predecessors = list(graph.predecessors(current))
        for pred in predecessors:
            if graph.nodes[pred]["Class_name"] == "HighVegetation":
                dfs(path + [pred])
        if not predecessors or all(graph.nodes[p]["Class_name"] != "HighVegetation" for p in predecessors):
            histories.append(path[::-1])
    if graph.nodes[central_node]["Class_name"] == "HighVegetation":
        dfs([central_node])
    else:
        print("Central node is not HighVegetation, no paths returned.")

    return histories

#Object of interest
node_of_interest = (1960, 2902)

histories_selected_centralnode = get_all_histories(G, node_of_interest)

# Print summary info
print(f"\nEnd object: {node_of_interest}, Class: {G.nodes[node_of_interest]['Class_name']}, Area:
{G.nodes[node_of_interest]['Area_in_m2']} m2, Paths: {len(histories_selected_centralnode)}")

# Print history details
for i, path in enumerate(histories_selected_centralnode):
    print(f"\nPath {i+1}:")
    for node in path:
        year, obj_id = node
        class_name = G.nodes[node]["Class_name"]
        growthtransitions = G.nodes[node].get("growth_type")
        degradationtransitions = G.nodes[node].get("degradation_type")

        if growthtransitions and degradationtransitions:
            transition_pattern = f"Growth: {growthtransitions}, Degradation: {degradationtransitions}"
        elif growthtransitions:
            transition_pattern = f"Growth: {growthtransitions}"
        elif degradationtransitions:
            transition_pattern = f"Degradation: {degradationtransitions}"
        else:
            transition_pattern = "Unknown"

        print(f"  {year},{obj_id}: Class_name = {class_name}, Transition pattern = {transition_pattern}")


#%%

#Object future choice for end object
def get_all_futures(graph, central_node):
    """Search for all paths from central_node to subsequent objects (multiple parents possible)"""
    futures = []

    def dfs(path):
        current = path[-1]
        successors = list(graph.successors(current))
```

```
        for succ in successors:
            if graph.nodes[succ]["Class_name"] == "HighVegetation":
                dfs(path + [succ])
        if not successors or all(graph.nodes[s]["Class_name"] != "HighVegetation" for s in successors):
            futures.append(path)

    if graph.nodes[central_node]["Class_name"] != "HighVegetation":
        return []  # Not valid start
    dfs([central_node])
    return futures

#Object of interest
node_of_interest = (1960, 2902)

futures_selected_centralnode = get_all_futures(G, node_of_interest)

# Print summary info
print(f"\nStart object: {node_of_interest}, Class: {G.nodes[node_of_interest]['Class_name']}, Area:
{G.nodes[node_of_interest]['Area_in_m2']} m2, Paths: {len(futures_selected_centralnode)}")

# Print history details
for i, path in enumerate(futures_selected_centralnode):
    print(f"\nPath {i+1}:")
    for node in path:
        year, obj_id = node
        class_name = G.nodes[node]["Class_name"]
        growthtransitions = G.nodes[node].get("growth_type")
        degradationtransitions = G.nodes[node].get("degradation_type")

        if growthtransitions and degradationtransitions:
            transition_pattern = f"Growth: {growthtransitions}, Degradation: {degradationtransitions}"
        elif growthtransitions:
            transition_pattern = f"Growth: {growthtransitions}"
        elif degradationtransitions:
            transition_pattern = f"Degradation: {degradationtransitions}"
        else:
            transition_pattern = "Unknown"

        print(f"  {year},{obj_id}: Class_name = {class_name}, Transition pattern = {transition_pattern}")

#%%
def get_full_lifecycles(graph, central_node):
    """Combine all history and future paths of a HighVegetation patch into full lifecycle paths."""
    if graph.nodes[central_node]["Class_name"] != "HighVegetation":
        print("Central node is not HighVegetation, lifecycle paths not applicable.")
        return []

    histories = get_all_histories(graph, central_node)
    futures = get_all_futures(graph, central_node)

    full_lifecycles = []
    for h in histories:
        for f in futures:
            if h[-1] == f[0]:  # Common node is central_node
                full_path = h + f[1:]  # Avoid duplicating the central node
                full_lifecycles.append(full_path)

    return full_lifecycles

node_of_interest = (1960, 2902)
lifecycles = get_full_lifecycles(G, node_of_interest)

print(f"\nLifecycle paths for object {node_of_interest}: {len(lifecycles)} total")
```

```
for i, path in enumerate(lifecycles):
    print(f"\nLifecycle {i+1}:")
    for node in path:
        year, obj_id = node
        class_name = G.nodes[node]["Class_name"]
        growthtransitions = G.nodes[node].get("growth_type")
        degradationtransitions = G.nodes[node].get("degradation_type")

        if growthtransitions and degradationtransitions:
            transition_pattern = f"Growth: {growthtransitions}, Degradation: {degradationtransitions}"
        elif growthtransitions:
            transition_pattern = f"Growth: {growthtransitions}"
        elif degradationtransitions:
            transition_pattern = f"Degradation: {degradationtransitions}"
        else:
            transition_pattern = "Unknown"

        print(f"  {year},{obj_id}: Class_name = {class_name}, Transition pattern = {transition_pattern}")

#%%

all_lifecycles = []

for node in G.nodes:
    if G.nodes[node]["Class_name"] == "HighVegetation":
        lifecycles = get_full_lifecycles(G, node)
        all_lifecycles.extend(lifecycles)

unique_lifecycles = []
seen = set()

for path in all_lifecycles:
    path_tuple = tuple(path)
    if path_tuple not in seen:
        seen.add(path_tuple)
        unique_lifecycles.append(path)

print(f"Total lifecycle paths found: {len(all_lifecycles)}")
print(f"Unique lifecycle paths after removing duplicates: {len(unique_lifecycles)}")

# Show 10 lifecycle examples
for i, path in enumerate(unique_lifecycles[:10]):
    print(f"\nLifecycle {i+1}:")
    for node in path:
        year, obj_id = node
        class_name = G.nodes[node]["Class_name"]
        growthtransitions = G.nodes[node].get("growth_type")
        degradationtransitions = G.nodes[node].get("degradation_type")

        if growthtransitions and degradationtransitions:
            transition_pattern = f"Growth: {growthtransitions}, Degradation: {degradationtransitions}"
        elif growthtransitions:
            transition_pattern = f"Growth: {growthtransitions}"
        elif degradationtransitions:
            transition_pattern = f"Degradation: {degradationtransitions}"
        else:
            transition_pattern = "Unknown"

        print(f"  {year},{obj_id}: Class_name = {class_name}, Transition pattern = {transition_pattern}")


lifecycle_durations = [len(path) for path in all_lifecycles]
```

```
duration_counts = Counter(lifecycle_durations)
duration_counts

#%%
### Plot Lifecycle Durations ###

# Extract start and end year for each unique lifecycle path
lifecycles_start_end = [(path[0][0], path[-1][0]) for path in unique_lifecycles]

# Count frequency of each (start_year, end_year) pair
freq_counts = Counter(lifecycles_start_end)

# Get list of unique lifecycle start/end pairs
unique_pairs = list(freq_counts.keys())

# Sort pairs by start year (for neat plotting)
unique_pairs = sorted(unique_pairs, key=lambda x: x[0])

filtered_pairs = [pair for pair in unique_pairs if freq_counts[pair] > 0]

# Normalize frequency for color mapping
max_freq = max(freq_counts.values())

cmap = plt.cm.viridis

fig, ax = plt.subplots(figsize=(12, 8))

# Plot one arrow per unique lifecycle (start -> end)
for i, (start, end) in enumerate(unique_pairs):
    freq = freq_counts[(start, end)]
    color = cmap(freq / max_freq)

    # Draw arrow on y=i from start to end
    if start == end:
        # Plot a dot for single-year lifecycle
        ax.plot(start, i, 'o', color=color, markersize=8)
    else:
        # Plot an arrow for multi-year lifecycle
        ax.annotate(
            '',
            xy=(end, i),
            xytext=(start, i),
            arrowprops=dict(arrowstyle='->', color=color, lw=3)
        )

# Show frequencies as y-axis labels
freq_labels = [str(freq_counts[pair]) for pair in unique_pairs]
ytick_labels = [f"{start} → {end}" for start, end in filtered_pairs]

ax.set_yticks(range(len(filtered_pairs)))
ax.set_yticklabels(ytick_labels, fontsize=8)
ax.set_ylabel('Lifecycle Start → End Year')


# Set x-axis limits explicitly to first and last year of all lifecycles
all_years = [year for pair in unique_pairs for year in pair]
xmin, xmax = min(all_years), max(all_years)
ax.set_xlim(xmin - 1, xmax + 1)
# Annotate all years on x-axis as ticks and labels
all_years_sorted = sorted(set(all_years))
ax.set_xticks(all_years_sorted)
ax.set_xticklabels(all_years_sorted, rotation=45, ha='right', fontsize=8)
ax.set_xlabel('Year')
```

```
ax.set_title('HighVegetation Lifecycles: Arrow from Start Year to End Year')
ax.set_ylim(-0.5, len(filtered_pairs) - 0.5)
# Colorbar for frequency
sm = plt.cm.ScalarMappable(cmap=cmap, norm=plt.Normalize(vmin=1, vmax=max_freq))
sm.set_array([])
cbar = plt.colorbar(sm, ax=ax, pad=0.02)
cbar.set_label('Frequency of lifecycle (start-end)')

plt.tight_layout()
plt.show()

#%%
### First HV in lifecycle growth transition ###
first_growth_transitions = []

for lifecycle in unique_lifecycles:
    if lifecycle:  # non-empty lifecycle
        first_node = lifecycle[0]
        growth_transition = G.nodes[first_node].get("growth_type")
        if growth_transition:
            first_growth_transitions.append(growth_transition)

# Count total occurrences of each growth transition type
growth_counts = Counter(first_growth_transitions)

print("Counts of growth transitions for the first HV node in each lifecycle:")
for transition, count in growth_counts.most_common():
    print(f"{transition}: {count}")

#%%
### Last HV in lifecycle degradation transition
last_degradation_transitions = []

for lifecycle in unique_lifecycles:
    if lifecycle:  # non-empty lifecycle
        last_node = lifecycle[-1]
        degradation_transition = G.nodes[last_node].get("degradation_type")
        if degradation_transition:
            last_degradation_transitions.append(degradation_transition)

# Count total occurrences of each growth transition type
degradation_counts = Counter(last_degradation_transitions)

print("Counts of degradation transitions for the last HV node in each lifecycle:")
for transition, count in degradation_counts.most_common():
    print(f"{transition}: {count}")

#%%
### Duration distribution ###
durations = [path[-1][0] - path[0][0] for path in unique_lifecycles]

plt.figure(figsize=(8, 4))
sns.histplot(durations, bins=range(0, max(durations)+2), kde=False)
plt.yscale('log')
plt.title("Frequency of Lifecycle Durations")
plt.xlabel("Duration (years)")
plt.ylabel("Number of Lifecycles")
plt.show()

#%%
### First year and End Year distribution ###

origin_years = [path[0][0] for path in unique_lifecycles]
```

```
end_years = [path[-1][0] for path in unique_lifecycles]

plt.figure(figsize=(10, 4))
sns.countplot(x=origin_years, order=sorted(set(origin_years)))
plt.title("Number of HV Patches Originating per Year")
plt.ylabel("Count")
plt.xlabel("Start Year")
plt.show()

plt.figure(figsize=(10, 4))
sns.countplot(x=end_years, order=sorted(set(end_years)))
plt.title("Number of HV Patches Ending per Year")
plt.ylabel("Count")
plt.xlabel("End Year")
plt.show()

#%%
### Transition type per step in lifecycle ###

growth_types_by_pos = []
degrad_types_by_pos = []

for path in unique_lifecycles:
    transitions = [G.nodes[node].get("growth_type") for node in path[:-1]]
    growth_types_by_pos.extend([(i, t) for i, t in enumerate(transitions)])

    transitions = [G.nodes[node].get("degradation_type") for node in path[1:]]
    degrad_types_by_pos.extend([(i, t) for i, t in enumerate(transitions)])

# -- Convert to DataFrame for plotting
growth_df = pd.DataFrame(growth_types_by_pos, columns=["Position", "Type"])
degrad_df = pd.DataFrame(degrad_types_by_pos, columns=["Position", "Type"])

# Plot growth types by position
plt.figure(figsize=(12, 6))
sns.countplot(data=growth_df, x="Position", hue="Type")
plt.title("Growth Transition Types by Lifecycle Position")
plt.xlabel("Lifecycle Position")
plt.ylabel("Frequency")
plt.legend(title="Growth Type", bbox_to_anchor=(1.05, 1), loc='upper left')
plt.tight_layout()
plt.show()

# Plot degradation types by position
plt.figure(figsize=(12, 6))
sns.countplot(data=degrad_df, x="Position", hue="Type")
plt.title("Degradation Transition Types by Lifecycle Position")
plt.xlabel("Lifecycle Position")
plt.ylabel("Frequency")
plt.legend(title="Degradation Type", bbox_to_anchor=(1.05, 1), loc='upper left')
plt.tight_layout()
plt.show()

#%%
### Transition type per age in lifecycle ###

growth_types_by_age = []
degrad_types_by_age = []

for path in unique_lifecycles:
    # Sort path by year (first element of tuple)
    sorted_path = sorted(path, key=lambda x: x[0])
    start_year = sorted_path[0][0]
```

```
    # Growth transitions: all nodes except last
    for node in sorted_path[:-1]:
        year = node[0]  # Year is the first item of the tuple
        age = year - start_year
        growth_type = G.nodes[node].get("growth_type")
        area = G.nodes[node].get("Area_t")
        if growth_type:
            growth_types_by_age.append((age, growth_type, area))

    # Degradation transitions: all nodes except first
    for node in sorted_path[1:]:
        year = node[0]
        age = year - start_year
        degrad_type = G.nodes[node].get("degradation_type")
        area = G.nodes[node].get("Area_t")
        if degrad_type:
            degrad_types_by_age.append((age, degrad_type, area))

# Convert to DataFrames
growth_df = pd.DataFrame(growth_types_by_age, columns=["Age", "Type", "Area"])
degrad_df = pd.DataFrame(degrad_types_by_age, columns=["Age", "Type", "Area"])

# Plot Growth
plt.figure(figsize=(12, 6))
sns.countplot(data=growth_df, x="Age", hue="Type")
plt.title("Growth Transition Types by Age (Years since lifecycle start)")
plt.xlabel("Lifecycle Age (Years)")
plt.ylabel("Frequency")
plt.legend(title="Growth Type", bbox_to_anchor=(1.05, 1), loc='upper left')
plt.tight_layout()
plt.show()

# Plot Degradation
plt.figure(figsize=(12, 6))
sns.countplot(data=degrad_df, x="Age", hue="Type")
plt.title("Degradation Transition Types by Age (Years since lifecycle start)")
plt.xlabel("Lifecycle Age (Years)")
plt.ylabel("Frequency")
plt.legend(title="Degradation Type", bbox_to_anchor=(1.05, 1), loc='upper left')
plt.tight_layout()
plt.show()
#%%
### Transition types by age as percentage of count ###

# --- Group: Count occurrences
growth_count_df = growth_df.groupby(["Age", "Type"]).size().reset_index(name="Count")
degrad_count_df = degrad_df.groupby(["Age", "Type"]).size().reset_index(name="Count")

# --- Pivot to get totals per Age and type
growth_pivot_freq = growth_count_df.pivot_table(index="Age", columns="Type", values="Count", fill_value=0)
degrad_pivot_freq = degrad_count_df.pivot_table(index="Age", columns="Type", values="Count", fill_value=0)

# --- Convert to percentages
growth_percent = growth_pivot_freq.div(growth_pivot_freq.sum(axis=1), axis=0) * 100
degrad_percent = degrad_pivot_freq.div(degrad_pivot_freq.sum(axis=1), axis=0) * 100

# --- Plot Growth Frequency (%)
plt.figure(figsize=(12, 6))
growth_percent.sort_index().plot(kind="bar", stacked=True, colormap="tab20", width=0.9)
plt.title("Growth Transition Types by Age (Percentage of Transitions)")
plt.xlabel("Lifecycle Age (Years)")
plt.ylabel("Percentage (%)")
```

```
plt.xticks(rotation=45)
plt.legend(title="Growth Type", bbox_to_anchor=(1.05, 1), loc='upper left')
plt.tight_layout()
plt.show()

# --- Plot Degradation Frequency (%)
plt.figure(figsize=(12, 6))
degrad_percent.sort_index().plot(kind="bar", stacked=True, colormap="tab20c", width=0.9)
plt.title("Degradation Transition Types by Age (Percentage of Transitions)")
plt.xlabel("Lifecycle Age (Years)")
plt.ylabel("Percentage (%)")
plt.xticks(rotation=45)
plt.legend(title="Degradation Type", bbox_to_anchor=(1.05, 1), loc='upper left')
plt.tight_layout()
plt.show()

#%%
### Transition types by age as percentage of area ###

growth_area_df = growth_df.groupby(["Age", "Type"])["Area"].sum().reset_index()
degrad_area_df = degrad_df.groupby(["Age", "Type"])["Area"].sum().reset_index()

# --- Pivot for stacked bar plotting
growth_pivot = growth_area_df.pivot_table(index="Age", columns="Type", values="Area", fill_value=0)
degrad_pivot = degrad_area_df.pivot_table(index="Age", columns="Type", values="Area", fill_value=0)

# --- Plot Growth Area
plt.figure(figsize=(12, 6))
growth_pivot.sort_index().plot(kind="bar", stacked=True, colormap="tab20", width=0.9)
plt.title("Growth Transition Types by Age (Total Area)")
plt.xlabel("Lifecycle Age (Years)")
plt.ylabel("Total Area (m²)")
plt.xticks(rotation=45)
plt.legend(title="Growth Type", bbox_to_anchor=(1.05, 1), loc='upper left')
plt.tight_layout()
plt.show()

# --- Plot Degradation Area
plt.figure(figsize=(12, 6))
degrad_pivot.sort_index().plot(kind="bar", stacked=True, colormap="tab20c", width=0.9)
plt.title("Degradation Transition Types by Age (Total Area)")
plt.xlabel("Lifecycle Age (Years)")
plt.ylabel("Total Area (m²)")
plt.xticks(rotation=45)
plt.legend(title="Degradation Type", bbox_to_anchor=(1.05, 1), loc='upper left')
plt.tight_layout()
plt.show()
#%%
import geopandas as gpd
from shapely.geometry import Point
import matplotlib.pyplot as plt
import rasterio
from rasterio.plot import show
import os

# Year list and raster folder
years = [1946, 1954, 1960, 1968, 1975, 1980, 1985, 1992, 1998, 2000, 2002, 2009, 2013, 2016, 2019]
raster_dir =
r"C:\Users\timme\OneDrive\Bureaublad\Master\Master_Thesis\rasterized_Full_classifications_subsets\subset3_output"

lifecycle_points = []

for path in unique_lifecycles:
```

```
    sorted_path = sorted(path, key=lambda x: x[0])
    start_year = sorted_path[0][0]
    end_year = sorted_path[-1][0]

    for i, node in enumerate(sorted_path):
        node_year, obj_id = node
        x = G.nodes[node].get("xcoor")
        y = G.nodes[node].get("ycoor")
        if x is None or y is None:
            continue

        if i == 0:
            # Start node: plot in actual start year
            plot_year = node_year
            point_type = "Start"
        elif i == len(sorted_path) - 1:
            # End node: plot in year AFTER end, only if it exists
            next_year_index = years.index(node_year) + 1 if node_year in years else None
            if next_year_index is not None and next_year_index < len(years):
                plot_year = years[next_year_index]
                point_type = "End"
            else:
                continue  # Skip if no valid "next" year in list
        else:
            # Intermediate node: plot in its own year
            plot_year = node_year
            point_type = "Intermediate"

        lifecycle_points.append({
            "geometry": Point(x, y),
            "Year": plot_year,
            "Object_ID": obj_id,
            "Type": point_type
        })

# Create GeoDataFrame
points_gdf = gpd.GeoDataFrame(lifecycle_points, geometry="geometry", crs="EPSG:32631")
# Use one of the rasters to get its transform
sample_raster =
"C:/Users/timme/OneDrive/Bureaublad/Master/Master_Thesis/rasterized_Full_classifications_subsets/subset3_output/clas
sified_2019.tif"
with rasterio.open(sample_raster) as src:
    transform = src.transform
    height = src.height
    width = src.width

# Invert Y values
def invert_y(y):
    # Use affine transform to find top-left origin and pixel size
    # In rasterio, y decreases down, so invert relative to height
    return transform.f + height * transform.e - (y - transform.f)

# Apply to your GeoDataFrame
points_gdf["geometry"] = points_gdf["geometry"].apply(
    lambda point: Point(point.x, invert_y(point.y))
)
raster_path = os.path.join(raster_dir, f"classified_{year}.tif")
# 3. Plot each year
for year in years:
    # Get matching raster file
    raster_path = os.path.join(raster_dir, f"classified_{year}.tif")

    # Filter lifecycle points for this year
```

```
year_points = points_gdf[points_gdf["Year"] == year]

# Skip if no points
if year_points.empty:
    print(f"No lifecycle points found for year {year}")
    continue

try:
    with rasterio.open(raster_path) as src:
        fig, ax = plt.subplots(figsize=(10, 10))
        show(src, ax=ax, cmap="terrain")

        # Plot points per type
        color_map = {"Start": "green", "Intermediate": "orange", "End": "red"}
        for point_type, color in color_map.items():
            subset = year_points[year_points["Type"] == point_type]
            if not subset.empty:
                subset.plot(ax=ax, color=color, markersize=100, label=point_type)

        plt.title(f"High Vegetation Lifecycle Locations – {year}")
        plt.xlabel("Easting (m)")
        plt.ylabel("Northing (m)")
        plt.legend()
        plt.tight_layout()

        # Optional: Save to file
        # plt.savefig(f"lifecycle_map_{year}.png", dpi=300)

        plt.show()
except FileNotFoundError:
    print(f"Raster not found for year {year}: {raster_path}")
```