Sen4CAP - Sentinels for Common Agricultural Policy

Design Justification File

ATBD for Parcels Heterogeneity Check

Milestone	CCN2 - Milestone 5
Authors	Diane HEYMANS, Sophie BONTEMPS, Pierre DEFOURNY, Laurentiu NICOLA, Cosmin UDROIU
Distribution	ESA - Zoltan SZANTOI

This page is intentionally left blank

Table of recorded changes

Version	Date	Reason
V0.1	13/01/2023	Internal version
v1.0	17/03/2023	First version delivered to ESA
v1.1	30/03/2023	Updated of the first version based on ESA comments

Table of contents

List of figures

List of Algorithm

List of tables

References

ID	Title
RD.1	Sen4CAP Design Definition File - ATBD for the Subsidy Application Layer Preparation, version 1.1, 30 March 2021
RD.2	Sen4CAP Design Definition File - ATBD for the Crop Type mapping, version 1.3, 1 April 2021

List of acronyms

Acronym	Definition
ATBD	Algorithm Theoretical Basis Document
GSAA	GeoSpatial Aid Application
LPIS	Land Parcel Identification System
LUT	Look-Up Table
NDVI	Normalized Difference Vegetation Index
NDWI	Normalized Difference Water Index
PA	Paying Agency
RF	Random Forest
S1	Sentinel-1
S2	Sentinel-2
SAR	Synthetic Aperture Radar
Sen2-Agri	Sentinel-2 for Agriculture
SMOTE	Synthetic Minority Over-Sampling Technique
SWIR	Short-Wave Infrared
UTM	Universal Transvere Mercator

1. Logical model – overview of the processor

The developed parcel heterogeneity module relies on a clustering algorithm, run on optical Sentinel-2 (S2) surface reflectance and Synthetic Aperture Radar (SAR) Sentinel-1 (S1) time series at the tile-level. Clusters are then interpreted at the parcel-level to decide if the parcel is heterogeneous or not.

Figure 1-1 presents the general workflow of the parcel heterogeneity detection algorithms. There are 3 main components:

1. Input data preparation:
    a. Declaration data (hereafter referred to as "subsidy application layer");
    b. Optical data;
    c. SAR data;
2. Clustering:
    a. Clustering algorithm;
    b. Spatial smoothing;
    c. Spatial connectivity;
3. Parcel-level analysis:
    a. Parcel-level statistics extraction;
    b. Parcel-level markers computation;
    c. Heteregeneity decision rules.

Optical and SAR data pre-processing is embedded in the Sen4CAP system. From the pre-processed data (Level 2A (L2A) time series from S2 and backscatter and coherence time series from S1), resampled data are created every 10 days for S2 and every 7 days for S1 in the raster format.

The clustering is then applied on those resampled data. Then, isolated pixels are removed using a spatial smoothing operator and the connectivity between clusters inside each parcel is calculated. The processor is designed to run all these steps at the tile-level and for each period P (defined by the user).

In the final component, varying statistics are derived from the clustering steps for each parcel and heterogeneity markers are derived for each period P. Heterogeneity decision rules are finally applied to decide if the parcel is heterogeneous or not by looking at the markers over successive periods P.

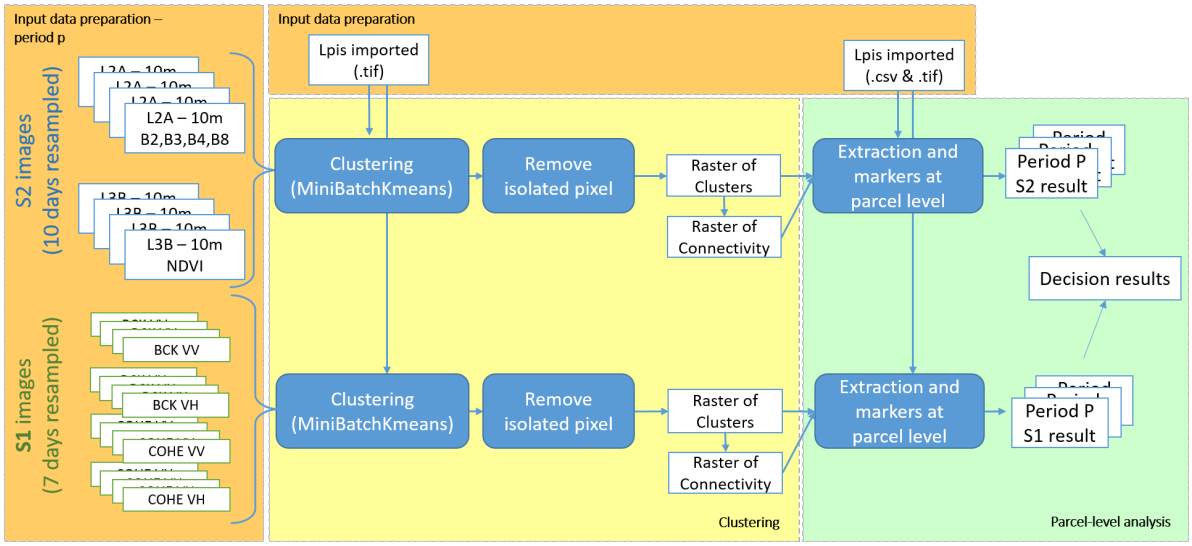

Figure 1-1. General workflow of the L4D heterogeneity detection algorithm

Figure 1-2 illustrates the periodicity of the general workflow. From the start of the season, the resampled images are produced along the season. At each period P (defined by the user), the clustering algorithm is run (yellow box from Figure 1-1) and the extraction of statistics and markers is run (first two steps of green box in Figure 1-1). Once at least three periods P are available, it is possible to create the first decision results. The decision result will evolve with the availability of new periods P. Figure 1-2 represents the workflow using S2 data, but the same applies for S1.

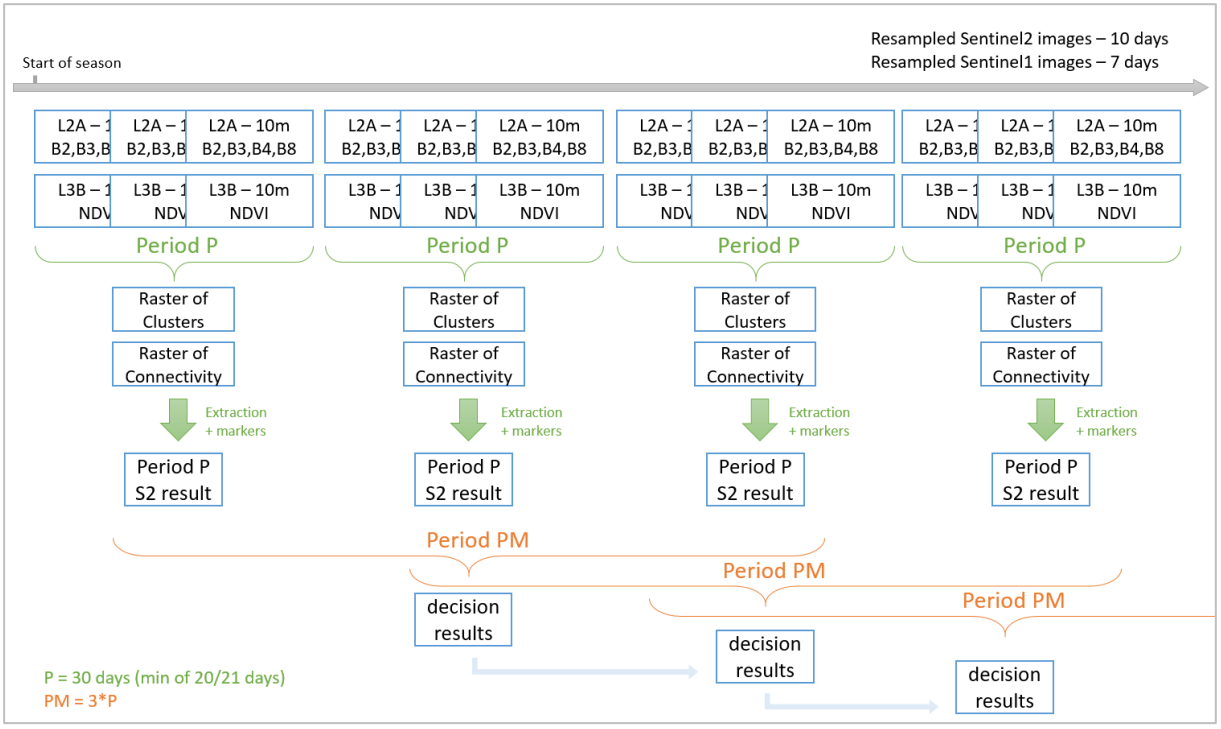

Figure 1-2. Period's representation of the general workflow of the L4D heterogeneity detection algorithm

In the following sections, the different steps are presented in details. For most of these steps, the specific input and output variables, as well as the code or pseudo-code, are given.

2. Input data preparation

2.1 Subsidy application layer

In order to ensure a certain level of consistency between the different Sen4CAP processors, the preparation of the subsidy application layer is performed prior to the execution of all processors. The subsidy application layer preparation is described in a dedicated ATBD (RD.1). The outputs of the subsidy application layer preparation that are used by the L4D parcels heterogeneity processor are described below.

2.1.1 Standardized subsidy application layer with quality flags

The standardized subsidy application layer with quality flags (Table 2-1):

- is stored as a PostGIS layer in the PostgreSQL database of the system;
- is projected in national projection;
- has the following name: decl_{site}_{year};
- has the same number of rows (parcels) than the original subsidy application layer.

Table 2-1. Standardized subsidy application layer with quality flags

Output data	Description	Default value [format]
decl_{site}_{year}	The standardized version of the subsidy application layer with the quality flags: geometry and spectral information	[GPKG] & [CSV]

It contains the attribute fields listed in Table 2-2 (fields in orange are already present in the original subsidy application layer). Attributes coming from the LUT (Table 2-4) are also available in the layer at the parcel level. This layer is available as .gpkg and .csv.

Table 2-2. Content of the standardized subsidy application layer with quality flags

Field name	Role	Default value [format]
Ori attributes	All the original attributes of the original delaration dataset	[integer, float or string]
ori_id	Copy of the content of the attribute field defined by the user with the parcel id	[string]
ori_hold	Copy of the content of the attribute field defined by the user with the holding id	[string]
ori_crop	Copy of the content of the attribute field defined by the user with the crop code	[input format: string or integer]
NewID	New sequential ID of the parcel	[integer]
HoldID	New sequential ID of the holdings	[integer]
GeomValid	Identify parcels for which no polygon exists in the subsidy application layer or with a not valid geometry	[integer, binary]
Duplic	Identify parcels that have the exact same geometry as another	[integer, binary]

Area_meters	Parcel area in the UTM projection (m²)	[integer]
Overlap	Identify parcels which overlaps with neighbouring parcels	[integer, binary]
ShapeInd	Shape index of the parcel	[float]
S1pix	Indicates the number of used S1 pixels in the parcel	[integer]
S2pix	Indicates the number of used S2 pixels in the parcel	[integer]

2.1.2 Parcels raster layers

The parcels raster layers are the rasters that are produced for both sensors (S2 and S1) and by tile, with the NewID as value. The NewID of the parcel has been assigned as value only for the pixels that have their centroid located within the parcels inner buffer layers of 5 m for S2 and 10 m for S1 (Figure 2-1). Only these pixels will be used to extract the S1 and S2 spectral values.

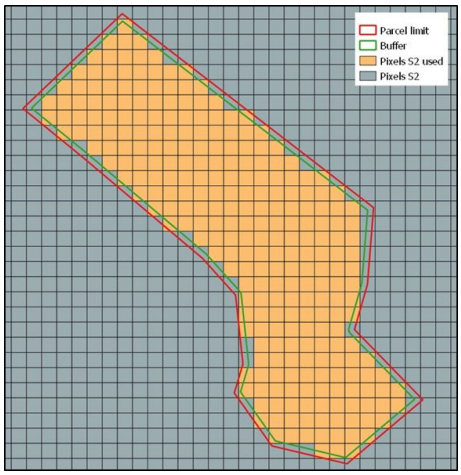

Figure 2-1. Selection of the used S2 pixels by parcel

These layers (Table 2-3):

- are .tif files;
- are produced by S2 tile;
- are projected in the WGS 84 / UTM zone {x} corresponding to the UTM zone of the S2 tile;
- have a value = NewID of the parcels;
- have a resolution of 10m for S2 and 20m for S1.

Table 2-3. Parcels raster layers

Output variable	Role	Default value [format]
decl_{site}_{year}_{ti}_S2	Raster of all the used S2 pixels by parcel (value as NewID); {ti} = name of the tile (ex. 31UFS)	[tif] (nr of tiles)
decl_{site}_{year}_{ti}_S1	Raster of all the used S1 pixels by parcel (value as NewID); {ti} = name of the tile (ex. 31UFS)	[tif] (nr of tiles)

2.1.3 Crop code LUT

If the original subsidy application layer contains a large number of crop types, it considerably improves the classification accuracy to group together the crop types that are by definition very similar or that

have a very similar phenology. It is done in the crop code Look-Up Table (LUT), which makes this grouping and defines new crop codes (CTnumL4A) and crop names (CTL4A).

In addition, to check the compliancy of the holdings regarding the crop diversification rules, a series of information should be defined by crop type: the crop diversification class (CTnumDIV and CTDIV) and whether or not it belongs to one or more of the categories Eligible Agricultural Area (EAA), Arable Land (AL), Permanent grassland, Temporary grassland, Fallow land and Crop under water.

All this information is summarized in a csv file, the crop code LUT, which:

- is stored as a table in the PostgreSQL database of the system;
- is named lut_{site}_{year};
- contains the following information (Table 2-4).

These attributes are also stored at the parcel level in the standardized subsidy application (decl_{site}_{year}).

Table 2-4. Content of the L4A crop code LUT

Field name	Role	Default value [format]
Ori_crop	The initial crop code from the subsidy application layer	[integer or string]
CTnum	The new crop type code (each Ori_crop being associated to a unique CTnum)	[integer]
CT	The name of the crop type in English	[string]
LC	The main land cover class of the crop type: ○ 0: other natural areas ○ 1: annual crop ○ 2: permanent crop ○ 3: grassland ○ 4: fallow land ○ 5: greenhouse and nursery	[integer]
CTnumL4A	The new crop type code resulting of the grouping of the CTnum for the classification	[integer]
CTL4A	The crop type name associated to CTnumL4A	[string]
CTnumDIV	The crop diversification class code	[integer]
CTDIV	The crop diversification class name	[string]
EAA	Eligble agricultural area: value 1 if the crop type belongs to this category, value 0 otherwise	[integer, binary]
AL	Arable Land: value 1 if the crop type belongs to this category, value 0 otherwise	[integer, binary]
PGrass	Permanent grassland: value 1 if the crop type belongs to this category, value 0 otherwise	[integer, binary]
TGrass	Temporary grassland: value 1 if the crop type belongs to this category, value 0 otherwise	[integer, binary]
Fallow	Fallow land: value 1 if the crop type belongs to this category, value 0 otherwise	[integer, binary]

Cwater	Crop under water: value 1 if the crop type belongs to this category, value 0 otherwise	[integer, binary]

2.2 Optical data

The optical data pre-processing is done in Sen4CAP, based on the MAJA algoritm. Only the bands at 10 meters resolution are used in this processor, as well as the Normalized Difference Vegetation Index (NDVI).

The optical data preparation corresponds, for these bands, to a temporal resampling and a gapfilling through linear interpolation of valid data at the pixel-level.

The objective of the temporal resampling is to generate a reflectance image time series which is gap-filled with respect to missing data and temporally resampled on a regular **10-day** grid. Missing data are refered to as the data masked as cloud, cloud shadow and saturated pixels. Implementation details are provided in a C++ script that can be launched as described in Algorithm 2-1. This algorithm should be launched independently for each S2 tile and for each S2 bands/variables.

Here are the inputs of the constant_step_interpolation_masked_i16 algorithm.

Table 2-5. Inputs and outputs of the constant_step_interpolation_masked_i16 algorithm

Input names	Role	Default value [format]
Input_file	The input file is a multiband raster (or vrt) containing all the images available for the period and for the S2 tile.	[tif] or [vrt] (nr of tiles*number of bands)
Mask_file	The mask file is a multiband raster (or vrt) that has the same size of the input_file but contains the cloud mask at 10m resolution.	[tif] or [vrt] (nr of tiles*number of bands)
Data_time	Data times corresponding to the input bands. (Eg. 15#22#25#32#35#42#45#52#52#55#62)	[float]
Period	Period for output bands times. Begin#interval#end. For a year long period with a 10 days interpolation: 0#10#360	[float]
No-masked values	Values of the mask_file corresponding to the valid observation. Val1#Val2#Val3 With MAJA algorithm: 0	0 [int]
Output_nan_value	Value of the nan in the output (optional)	-10000
max_dist	Maximum distance between two valid observations in days. If a hole length is > max_dist, it remains a hole	30 [int]
Output name	**Role**	**Default value [format]**
New_output_file	Name of the output_file containing the multiband raster with a constant step interpolation masked.	[tif] or [vrt] (nr of tiles*number of bands)

Algorithm 2-1. Temporal resampling of S2 time series algorithm

```
/usr/bin/constant_step_interpolation_masked_i16                /path_vrt/S2MAJA454_31UFS_2021_NDVI_2.vrt
/path_vrt_mask/S2MAJA454_31UFS_2021_MaskNDVI5.vrt    /path_out/S2MAJA454_31UFS_2021_NDVIMResampled4.tif
2#5#7#10#12#15#17#20#22#25#27#30#32#35#37#40#42#45#47#50#52#55#57#60#62#65#67#70#72#75#77#80#82#85#87#9
0#92#95#97#100#102#105#107#110#112#115#117#120#122#125#127#132#135#137#140#142#145#147#150#152#155#157#
160#162#165#167#170#172#175#177#180#182#185#187#190#192#195#197#200#202#205#207#210#212#215#217#220#222
#225#227#230#232#235#237#240#242#245#247#250#252#255#257#260#262#265#267#270#272#275#277#280#282#285#28
7#290#292#295#297#300#302#305#307#310#315#317#320#322#325#327#330#332#335#337#340#342#345#350#352#355#3
57#360#362#365 0#10#360 0
```

2.3 SAR data

The SAR data pre-processing is done in Sen4CAP and results in backscatter and coherence time series. The parcels heterogeneity algorithm relies on the backscatter VV, backscatter VH, coherence VV and coherence VH **weekly (7 days)** time series. The computation of these weekly mosaics is already implemented in the crop type L4A processor of the Sen4CAP system (RD.2).

2.3.1 Mosaicking and formatting

The method is designed to look for all the S1 images overlapping a given S2 tile within a given period. It produces time series of S1 mosaics combining images acquired within 7-day period. This step produces a temporally regular time series of mosaics fully covered by valid pixel (Figure 2-2). For classification purposes, the mosaics shall not be covered by no data pixels.

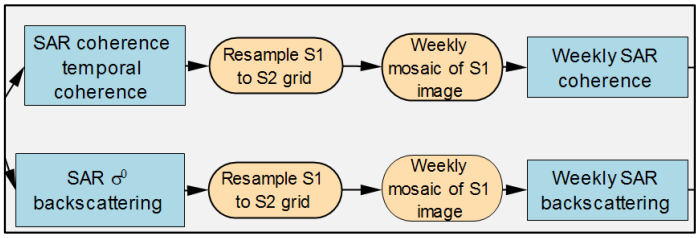

Figure 2-2.Workflow of the mosaicking step to produce weekly mosaics of S1 images

The mosaics are formated to match the format (resolution, extent and projection) of the S2 image given as input. The weekly mosaics are generated for each calendar week, this way, the mosaics are independent of the season duration.

The mosaicking step is performed independently for the two passes – ascending and descending – and for the two polarimetry – VV and VH – respectively for the coherence and the backscattering. It therefore results in 8 independent time series of

1. Coherence – Ascending – VV
2. Coherence – Ascending – VH
3. Coherence – Descending – VV
4. Coherence – Descending – VH
5. Backscattering intensity – Ascending – VV
6. Backscattering intensity – Ascending – VH
7. Backscattering intensity – Descending – VV
8. Backscattering intensity – Descending – VH

2.3.2 Gap-filling consideration

At the time of the processing chain design, no data filtering was applied on the SAR time series. The added value of applying a temporal resampling was therefore not highlighted. Only in the case of missing acquisitions, a gap-filling step must be applied.

3. Clustering algorithm

The clustering algorithm is run for Sentinel-1 and Sentinel-2 in parallel in two separate processing chains. For each S2 or S1 tile (S1 tile corresponding to S1 data clipped over the S2 tile extent) and for each period P, the clustering algorithm is run at the pixel-level and generates one raster of clusters.

3.1 Inputs and preparation of clustering

Before running the clustering algorithm, S1 and S2 images over the period P are imported and all pixels corresponding to non-crop areas and no data are masked using the declaration dataset imported and rasterized as described in section 2.1.2.

3.1.1 Input data

Table 3-1 describes the inputs of the clustering algorithm.

Table 3-1. Inputs of the clustering algorithm

Input names	Role	Default value [format]
decl_{site}_{year}_{ti}_S2 or decl_{site}_{year}_{ti}_S1	Raster of all the used S2/S1 pixels by parcel (value as NewID); {ti} = name of the tile (ex. 31UFS)	[tif] (nr of tiles)
site	site of the sen4cap system related to the AOI.	[string]
imG_path	Path to the resampled images of the period *p* and for the band_list.	[string]
P	Period on which the clustering is done. Minimum of 20days (2 resampled S2 images) or 21 days (3 resampled S1 images). Ex. Month of march (30 days – 3 resampled S2 images and 4 resampled S1 images)	30 [days]
band_list	List of the bands (variables) that will be used in the clustering. The default for S2: [B2, B3, B4, B8, NDVI] The default for S1: [Cohe_Asc_VV, Cohe_Asc_VH, Cohe_Desc_VV, Cohe_Desc_VH, Amp_Asc_VV, Amp_Asc_VH, Amp _Desc_VV, Amp _Desc_VH]	S2 list. S1 list. [list of string]
n_cl	Number of clusters. Varies according to the landscape. By default, this number is 4 in the S2 clustering and 5 in the S1 clustering of more hilly area.	4 when S2, 5 when S1 [int]

3.1.2 Import

The step consists in creating a stack of S2 resampled images and S1 weekly mosaics over the period *P*. The images of the period *P* are retrieved from the folder where they are stored in the *imG_path*.

The *img* will store the information of the bands of the *band_list* and all dates of the *date_list*. The *n_var* is the total numbe of variable (band_list * date_list). The *img* dimension is (x, y, *n_var*) with x, y being the size of the rater (number of pixels in row, column).

Algorithm 3-1. Creation of images stacks over the period *P*

```
    img = np.zeros((img_ds.RasterYSize,img_ds.RasterXSize,n_var),
                gdal_array.GDALTypeCodeToNumericTypeCode(img_ds.GetRasterBand(1).DataType))
    for d in range(len(date_list)):
        date = date_l[d]
        for b in range(len(band_list)):
            bands_n = band_list[b]
            All_images = glob.glob(f'{imG_path}/*_{bands_n}.tif')
            img_ds = gdal.Open(All_images[0],gdal.GA_ReadOnly)
            img_dsB = img_ds.GetRasterBand(date)
            img_dsA = img_dsB.ReadAsArray()
            img[:, :,(d*len(band_list))+b] = img_dsA
```

3.1.3 Masking non-crop and no-data values

Once the S1 and S2 images are uploaded into a stack of image (*img*), they need to be masked to remove non-cropland areas and in the case of S2, the no data. This is done using the "parcels raster layer" (see section 2.1.2) named decl_{site}_{year}_{i}_S2 or decl_{site}_{year}_{i}_S1. The values of these raster files correspond to the NewID values of the subsidy layer after the application of the inner buffer inside each parcel.

In these raster files, all pixels outside of the agricultural parcels are therefore coded as "0". In this step, only the pixels with values different from 0 are kept, thus removing all non-cropland areas. For S2, the no-data value (-10 000 and 0) are also set as "nan".

Algorithm 3-2. Masking non crop and no data pixels

```
    img0 = img[imgR!=0,:]
    img0 = img0.astype(float)
if satellite_id == 'S2':
        img0[img0==-10000] = np.nan
    img0[img0==0] = np.nan
```

3.2 Clustering

The clustering is done through a python function called "kmeans_missing", which has the specificity to be able to handle the no-data. The output of the clustering is then reshaped and saved.

3.2.1 kmeans_missing function

Table 3-2 lists the arguments and outputs of the kmeans_missing function.

Table 3-2. List of arguments and outputs of the kmeans_missing function

Arguments	Role	Default value [format]
X	An [n_samples, n_features] array of data to cluster	Array of n_var dimensions
n_cl	Number of clusters to be used in the KMeans algorithm	n_cl [int] - see value in Table 3-1

max_iter	Maximum number of iterations allowed for the algorithm to converge	10 [int]
Outputs	**Role**	**Default value [format]**
Labels	An [n_samples] vector of integer labels	Array of 1 dimension
Centroids	An [n_cl, n_features] array of cluster centroids	Vector of n_var length
X_hat	Copy of X with the missing values filled in	Array of n_var dimensions

This function allows running a KMeans algorithm, while filling in missing values by using the cluster centroids value:

- The function looks first for missing values in the array and stores this information in a new array called *X_hat*;
- Then, a first iteration is started (using the function MinibatchKMeans) using an array where the missing value are replaced by "0";
- After this iteration, the "0" corresponding to the missing values are replaced by the value of the centroids of the clusters to which they belong;
- Iterations continue, and each time, the new centroid values are recorded instead of the missing values;
- When the labels have stopped changing between successive iterations, it means there is a convergence and the algorithm stops.

The pseudo-code of this Kmeans_missing function is given in Algorithm 3-3.

Algorithm 3-3. Kmeans_missing function pseudo-code

```
def kmeans_missing(X, n_cl, max_iter=10):
    # Initialize missing values to their column means
    missing = ~np.isfinite(X)
    mu = np.nanmean(X, 0, keepdims=1)
    X_hat = np.where(missing, mu, X)
    for i in range(max_iter):
        if i > 0:
            # initialize KMeans with the previous set of centroids. this is much
            # faster and makes it easier to check convergence (since labels
            # won't be permuted on every iteration), but might be more prone to
            # getting stuck in local minima.
            cls = cluster.MiniBatchKMeans(n_cl, init=prev_centroids)
        else:
            # do multiple random initializations in parallel
            cls = cluster.MiniBatchKMeans(n_cl)
        # perform clustering on the filled-in data
        labels = cls.fit_predict(X_hat)
        centroids = cls.cluster_centers_
        # fill in the missing values based on their cluster centroids
        X_hat[missing] = centroids[labels][missing]
        # when the labels have stopped changing then we have converged
        if i > 0 and np.all(labels == prev_labels):
            break

        prev_labels = labels
        prev_centroids = cls.cluster_centers_
    return labels, centroids, X_hat
```

3.2.2 Image reconstruction and save temporary cluster raster

The clusters labels are retrieved and stored into the *t_cluster* variable. 1 is added to all the labels to avoid having a cluster with the value 0.

Then, the labels are written in the "parcels raster layers" (decl_{site}_{year}_{ti}_S2 and decl_{site}_{year}_{ti}_S1) where the values are different from 0. where the values were not equal to 0. The raster is finally reshaped into a matrix (*t_clusterE*) and then saved to the geotiff format (using the write_geotiff function), as shown in the Algorithms 3-4 and 3-5.

The temporary cluster files (for S1 and S2) are named ***L4D_Cl{S1 or S2}_{site}_{ti}_{p}_tmp.tif.***

Algorithm 3-4. Saving the cluster outputs into geotiff

```
def write_geotiff(filename, arr, in_ds):
    if arr.dtype == np.float32:
        arr_type = gdal.GDT_Float32
    else:
        arr_type = gdal.GDT_Int32

    driver = gdal.GetDriverByName("GTiff")
    out_ds = driver.Create(filename, arr.shape[1], arr.shape[0], 1, arr_type)
    out_ds.SetProjection(in_ds.GetProjection())
    out_ds.SetGeoTransform(in_ds.GetGeoTransform())
    band = out_ds.GetRasterBand(1)
    band.WriteArray(arr)
    band.FlushCache()
    band.ComputeStatistics(False)
```

Algorithm 3-5. Cluster images reconstruction

```
#Open raster of mask from the data preparation
raster = gdal.Open(raster_5mBuffer,gdal.GA_ReadOnly)
imgR = raster.ReadAsArray()
newshpR = (imgR.shape[0]*imgR.shape[1])
imgR0  = imgR[:,:].reshape(newshpR)
#retrieve the labels from the kmeans_missing function
t_cluster = t_missing[0]
X_clusterOut = t_cluster +1 #1 added to avoir 0 values
#get the label value in the mask layer reshaped
imgR0[imgR0!=0]=X_clusterOut
t_clusterE = imgR0.reshape(img_dsA.shape) #in matrix form
write_geotiff(file_cluster,t_clusterE,raster)
```

3.3 Spatial smoothing

A spatial filter is applied on the cluster outputs in order to reduce the number of isolated pixels. This spatial smoothing is based on the "remove_isolated" C++ function.

The "remove_isolated" function needs to define six inputs: the input and output files (that are in the tiff format), the list of class, the no-data value, the radius of the window in which looking for isolated pixels and the threshold to be considered as isolated and to be removed and replaced (Table 3-3).

Table 3-3. List of arguments and outputs of the remove_isolated function

Arguments	Role	Default value [format]
File_in	Temporary raster of cluster – one by tile	[tif] (nbr of tile)
List_class	List of classes (comma-separated without space). Depend on the number of clusters – range from 1 to n_cl+1	1,2,3,4,5 [string] (cluster label)

noDataClassif	No-data value	0 [int]
Radius	Radius of search (window of 3x3 pixels when radius = 1)	1 [int]
Min_nb	Isolated pixel threshold	1 [int]
Output	**Role**	**Default value [format]**
File_out_post	Raster of cluster without isolated pixel named L4D_Cl{S1 or S2}_{site}_{ti}_{p}.tif – one by tile *(ti)*	[tif] (nbr of tile)

The function removes the isolated pixels when the class it belongs to is weakly present in the search window, i.e. when the number of pixels having the same label is below the threshold "Min_nb", or if the pixel is labelled as no-data. In order to replace the removed pixels, the majority of the neighbour values within the search window is taken. The script is in C++ and can be launched as shown in Algorithm 3-6.

Algorithm 3-6. Spatial smoothing algorithm

```
def remove_isolated(file_in, file_out, noDataClassif, min_nb=3, radius=1):
    cmd = '/pathtoscript/removeIsolated'
    cmd += f' {file_out}'
    cmd += f' {file_in}'
    cmd += f' {min_nb}'
    cmd += f' {radius}'
    cmd += f' {noDataClassif}'
    subprocess.check_call(cmd, shell=True)
```

3.4 Spatial connectivity

This step aims at calculating the spatial connectivity at the parcel-level. This spatial connectivity will be used as a proxy for the compactness when deriving markers for the parcels' heterogeneity (section 0).

This step uses as input the output of the spatial smooting, i.e. a raster of clusters without isolated pixels. It generates a new raster file describing the connectivity using a function named "LocalClassConnectivityIndex", coded in C++.

Table 3-4 lists the arguments and outputs of this LocalClassConnectivityIndex function while the script is shown in Algorithm 3-7.

Table 3-4 List of arguments and outputs of the LocalClassConnectivityIndex function

Arguments	Role	Default value [format]
File_out_post	Raster of cluster after the removal of isolated pixels – one by tile (ex. L4D_Cl{S1 or S2}_{site}_{ti}_{p}.tif)	[tif] (nbr of tile)
RadiusC	Radius of search (window of 3x3 pixels when radius = 1)	1 [int]

Output	Role	Default value [format]
File_out_post2	Raster of connectivity named (L4D_Cl{S1 or S2}_{site}_{ti}_{p}_Connect.tif) – one by tile (ti)	[tif] (nbr of tile)

For each pixel within the parcel, the function counts in the radius of search (*RadiusC*) the number of pixels having the same label (i.e. belonging to the same cluter). A pixel having different clusters in this *RadiusC* will have low values of connectivity while a pixel surrounded by the same cluster will have high values.

Algorithm 3-7. Spatial connectivity function

```
def localConnectivity(file_in,file_out,radiusC):
    cmd = '/pathtoscript/localClassConnectivityIndex'
    cmd += f' {file_out_post1}'
    cmd += f' {file_out_post}'
    cmd += f' {radiusC}'
    #print(cmd)
    subprocess.check_call(cmd, shell=True)
```

4. Parcel-level analysis

The analysis at the parcel-level is done in three steps. First, statistics are extracted at the parcel-level for each cluster generated in the section 3 and so, for each period *P*. Second, these statistics are used to generate heterogeneity markers, still at the parcel-level and for each period *P*. Finally, these markers are analyzed over a longer period *PM* (corresponding to at least 3 periods *P*) to support the decision of heterogeneity following the traffic light approach and providing confidence levels (see Figure 1-2).

4.1 Parcel-level statistics extraction

The parcel-level stastics extraction is done for each cluster produced in the clustering component (see section 3). It differs sligthy for S1 and S2. Both workflows are explained below.

4.1.1 Inputs of the parcel values extraction

For each period *P*, which correspond to one clusters' raster file produced, and for each S2/S1 tile (*ti*), a series of files/parameters are needed for the extraction. These inputs are described in Table 4-1 for S2 and in Table 4-2 for S1.

Table 4-1. Inputs of the parcel values extraction algorithm for S2

Input names	Role	Default value [format]
decl_{site}_{year}_{ti}_S2	Raster of all the used S2 pixels by parcel (value as NewID); {ti} = name of the tile (ex. 31UFS)	[tif] (as many as existing tiles)
decl_{site}_{year}	Parcel declaration after the preparation by the Sen4CAP system	[csv]
File_out_post	Raster of clusters without isolated pixels named ***L4D_ClS2_{site}_{ti}_{p}.tif*** – one by tile (ti)	[tif] (as many as existing tiles)
File_out_post2	Raster of connectivity named ***L4D_ClS2_{site}_{ti}_{p}_Connect.tif*** – one by tile (ti)	[tif] (nbr of tile)
Image_NDVI	List of images during the p period of NDVI.	[string]
Site	Site of the sen4cap system	[string]
P	Period on which the clustering is done. Minimum of 20 days (2 resampled S2 images) E.g. March = 30 days corresponding to 3 resampled S2 images	30 [days]
n_im_p	Number of images from the resampling needed to complete the period.	3 [int]
n_cl	Number of clusters	n_cl [int] - see value in Table 3-1
PerHetero	Pixels percentage corresponding to the biggest cluster in the parcel. If the biggest cluster is above PerHetero, the parcel is considered as homogeneous.	0.9 [float]

NPixClS2	Minimum number of pixels in a cluster to compute the distNDVI.	20 [int]
Outputs	**Description**	**Default value [format]**
csv_out	Table containing the extraction at the parcel level of the cluster's statistics, named ***L4D_HeteS2_{site}_{year}_{p}.csv***	[csv]

Table 4-2. Inputs of the parcel values extraction algorithm for S1

Input names	Role	Default value [format]
decl_{site}_{year}_{ti}_S1	Raster of all the used S1 pixels by parcel (value as NewID); {ti} = name of the tile (ex. 31UFS)	[tif] (as many as existing tiles)
decl_{site}_{year}	Parcel declaration after the preparation by the Sen4CAP system	[csv]
File_out_post	Raster of clusters without isolated pixels named ***L4D_ClS1_{site}_{ti}_{P}.tif*** – one by tile (ti)	[tif] (as many as existing tiles)
File_out_post2	Raster of connectivity named: ***L4D_ClS1_{site}_{ti}_{p}_Connect.tif*** – one by tile (ti)	[tif] (nbr of tile)
Site	Site of the sen4cap system	[string]
p	Period on which the clustering is done. Minimum of 20 days (2 resampled S2 images) E.g. March = 30 days corresponding to 3 resampled S2 images	30 [days]
n_cl	Number of clusters	n_cl [int] - see value in Table 3-1
PerHetero	Pixels percentage corresponding to the biggest cluster in the parcel. If the biggest cluster is above PerHetero, the parcel is considered as homogeneous.	0.9 [float]
Outputs	**Description**	**Default value [format]**
csv_out	Table containing the extraction at the parcel level of the cluster statistics, named ***L4D_HeteS1_{site}_{year}_{p}.csv***	[csv]

4.1.2 Statistics extraction at the parcel-level for S2 analysis

Before the extraction, the NDVI raster files corresponding to the period *P* are imported in the environment (Algorithm 4-1). An array (*img*) having the size of (x, y, len(date_l)) with x, y being the number of pixels in the rows and columns of the NDVI images and len(date_l) being the number of resampled images used in the clustering of the period *P* (see date_list in section 3.1.2). The l_var is a list that will be used to identify the date of NDVI later in the extraction step.

Algorithm 4-1. NDVI import before the statistics extraction with S2

```
img = np.zeros((img_ds.RasterXSize,img_ds.RasterYSize,len(date_l)),
               gdal_array.GDALTypeCodeToNumericTypeCode(img_ds.GetRasterBand(1).DataType))
l_var = []
for d in range(len(date_l)):
    date = date_l[d]
    img_dsB = img_ds.GetRasterBand(date)
    img_dsA = img_dsB.ReadAsArray()
    img[:, :,d] = img_dsA
    l_var.insert(d,f'{bandextracted}_{date}')
l_var.insert(0,'label')
```

Next, the following raster files related to the period of analysis *P* and the tile (*ti*) are imported in the environment:

- decl_{site}_{year}_{ti}_S2.tif is imported as *imR*;
- file_out_post is imported as *Cluster_extract*;
- file_out_post2 is imported as *Cluster_connectE*.

A stack (*imgC*) is done with these three raster files (which should all have the same size (x, y) and the same size as the *img* previously imported (see Algorithm 4-1).

Then a loop on each parcel (*i*) of the tile (*ti*) is performed to extract a set of statistics using a Python loop (Algorithm 4-2) which runs as follows.

First, the position of the parcel *i* is identified using the *imgR* in the *imgC* stack and stored into *cl_i*. Using this position (*cl_i*), it extracts the values from the cluster using the *cluster_extact* in *imgC* and stores it into *val_poly* variable. This step allows calculating the number of pixels in each cluster *c* (***cl_{c}***) and its related percentage (***clP_{c}***).

Then, if the maximum percentage of a cluster is lower than *PerHetero* (0.9), the parcel is flagged as potentially heterogeneous (***Hete*** = 1) and is further analysed. Otherwise, ***Hete*** is set to 0 and the analysis stops there; the script moves to the next parcel.

In case of ***Hete*** = 1, the script continues as such:

- ❖ It extracts the values of the connectivity (from the stack *imgC*) at the position of the parcel (*cl_i*) and computes the mean of the connectivity values associated to all pixels (ouput ***Compact***);
- ❖ Another variable generated is the **CompactA** that takes the mean of the connectivity values associated to all pixels normalized by the log of the area of the parcel.
- ❖ From the *img*, the NDVI values are extracted at the position of the parcel (*cl_i*). The no-data values (-10000) are masked. The number of no-data on the parcel allow to generate two flags.
  - **HoleS2** is the sum of images in the period p for the parcel that is fully covered by NA (maximum is n_im_p);
  - **HoleS2Part** is the sum of images in the period p for the parcel that is partially covered by NA (maximum is n_im_p).
- ❖ The distances between each cluster that include more pixels than *NPixClS2* are calculated by taking the difference between the mean of the two clusters in absolute. From all the differences computed (*dist*) only the maximum of these differences is kept and stored into the ***distNDVI***.

Algorithm 4-2. Cluster_extract python function

```
df_cl = pd.DataFrame(columns=['NewID','Hete'])
for i in tqdm(newid_l):
    cl_i = np.where(imgC[0] == i)
    val_poly = imgC[1,cl_i[0],cl_i[1]]
```

```
    val_poly1 = val_poly[val_poly!=0]
    v_countP = np.bincount(val_poly1)/len(val_poly1)
    v_count = np.bincount(val_poly1)
    df_cl1 = pd.DataFrame({'NewID':[i]})
    df_cl1['ShapeInd'] = lpis_csv['ShapeInd'][i]
    df_cl1['LC'] = lpis_csv['lc'][i]
    if sum(val_poly)>0 :
        for c in range(1,max(val_poly1)+1):
            df_cl1[f'clP_{c}'] = v_countP[c]
            df_cl1[f'cl_{c}'] = v_count[c]

    if len(v_count) == 0:
        df_cl1['Hete'] = 2
        df_cl1['distNDVI'] = 0

    elif max(v_countP) < PerHetero:
        val_Connect = imgC[2,cl_i[0],cl_i[1]]
        valAll_poly = img[cl_i[0],cl_i[1],:]
        valAll_poly = np.concatenate([val_poly[:, np.newaxis],valAll_poly],axis=1)
        valAll_polyM = ma.array(valAll_poly,mask=[valAll_poly==-10000])
        cluster_i = np.unique(val_poly)

        dist = pd.DataFrame(columns=['cl_name','distNDVI'])
        valAll_poly_d = pd.DataFrame(valAll_polyM,columns=ndvi_col)

        for ii in range(1,max(cluster_i)+1):
            for j in reversed(range(1,max(cluster_i)+1)):
                if j<=ii:
                    continue
                else:
                    dist1 = pd.DataFrame({'cl_name': [str(ii)+','+str(j)]})
                    #print(str(i)+','+str(j))
                    #print(v_count[ii])
                    if (v_count[ii]>NPixTHRNDVI) & (v_count[j]>NPixTHRNDVI):
                            dist1['distNDVI'] = np.absolute(np.nanmean(m_polydG[m_polydG.label==ii]) -
np.nanmean(m_polydG[m_polydG.label==j]))
                    else:
                        dist1['distNDVI'] = 0
                    dist = pd.concat([dist,dist1])

        df_cl1['Compact'] = np.nanmean(val_Connect)
        df_cl1['CompactA'] = (np.nanmean(val_Connect)/np.log(lpis_csv[area][i]))

        df_cl1['Hete'] = 1
        df_cl1['distNDVI'] = round(np.nanmax(dist['distNDVI'])/1000,5)
    else:
        dist = pd.DataFrame()
        df_cl1['Hete'] = 0
        df_cl1['distNDVI'] = 0
    df_cl = pd.concat([df_cl,df_cl1])
df_cl = df_cl.fillna(0)
```

4.1.3 Statistics extraction at the parcel-level for S1 analysis

This step is similar to the statistics extraction at the parcel-level for S2 except the computation of the distNDVI which required the import of NDVI raster.

The following raster files related to the period of analysis *P* and the tile (*ti*) are imported in the environment:

- decl_{site}_{year}_{ti}_S1.tif is imported as *imR*;
- file_out_post is imported as *Cluster_extract*;
- file_out_post2 is imported as *Cluster_connectE*.

A stack (*imgC*) is done with these three raster files (which should all have the same size (x, y)).

Then a loop on each parcel (*i*) of the tile (*ti*) is performed to extract a set of statistics using a Python loop (Algorithm 4-2) which runs as follows.

First, the position of the parcel *i* is identified using the *imgR* in the *imgC* stack and stored into *cl_i*. Using this position (*cl_i*), it extracts the values from the cluster using the *cluster_extact* in *imgC* and stores it into *val_poly* variable. This step allows calculating the number of pixels in each cluster *c* (***cl_{c}***) and its related percentage (***clP_{c}***).

Then, if the maximum percentage of a cluster is lower than *PerHetero* (0.9), the parcel is flagged as potentially heterogeneous (***Hete*** = 1) and is further analysed. Otherwise, ***Hete*** is set to 0 and the analysis stops there; the script moves to the next parcel.

In case of ***Hete*** = 1, the script continues as such:

- It extracts the values of the connectivity (from the stack *imgC*) at the position of the parcel (*cl_i*) and computes the mean of the connectivity values associated to all pixels (ouput ***Compact***).
- Another variable generated is the **CompactA** that takes the mean of the connectivity values associated to all pixels normalized by the log of the area of the parcel.

Algorithm 4-3. Extraction cluster values python loop

```
df_cl = pd.DataFrame(columns=['NewID','Hete'])
for i in tqdm(newid):
    cl_i = np.where(imgC[0] == i)
    val_poly = imgC[1,cl_i[0],cl_i[1]]
    val_poly1 = val_poly[val_poly!=0]
    v_countP = np.bincount(val_poly1)/len(val_poly1)
    v_count = np.bincount(val_poly1)
    df_cl1 = pd.DataFrame({'NewID':[i]})
    if sum(val_poly)>0 :
        for c in range(1,max(val_poly1)+1):
            df_cl1[f'clP_{c}'] = v_countP[c]
            df_cl1[f'cl_{c}'] = v_count[c]
    if len(v_count) == 0:
        df_cl1['Hete'] = 0
    elif max(v_countP) < PerHetero:
        val_Connect = imgC[2,cl_i[0],cl_i[1]]
        df_cl1['Compact'] = np.nanmean(val_Connect)
        df_cl1['CompactA'] = np.nanmean(val_Connect)/ np.log(lpis_csv[area][i]))

        df_cl1['Hete'] = 1
    else:
        dist = pd.DataFrame()
        df_cl1['Hete'] = 0
    df_cl = pd.concat([df_cl,df_cl1])
df_cl = df_cl.fillna(0)
```

4.2 Parcel-level markers

According to the values of the statistics extracted in section 4.1, heterogeneity markers can be determined for each period *P*.

4.2.1 Parameters of the markers

Table 4-3 lists the parameters and their default values used to determine the markers observed for each period *P*.

Table 4-3. Parameters of heterogeneity markers

Parameters names	Role	Default value [format]
PerHetero	Pixels percentage corresponding to the biggest cluster in the parcel. If the biggest cluster is above PerHetero, the parcel is considered as homogeneous. Already used in the previous section (4.1.1)	0.9 [float]
NPixClS2	Minimum number of S2 pixels needed to take into consideration the cluster	20 [int]
NPixClS1	Minimum number of S1 pixels needed to take into consideration the cluster	20 [int]
ThrNDVIdist	Threshold of the NDVI distance calculated between clusters	0.17 [float]
ThrCompactS2	Threshold of the compactness in the S2 analysis (varies according to the radiusC which has a default value = 3)	3 [int]
ThrCompactS1	Threshold of the compactness in the S1 analysis (varies according to the radiusC which has a default value = 3)	3 [int]

4.2.2 Markers of heterogeneity

Heterogeneity markers are defined for S2 and S1 separately. They are presented in Table 4-4

Table 4-4. Heterogeneity markers for S2 and S1 calculated over each period *P*

Marker S2	Description	Possible Value
M1	More than one big cluster (>*PerHetero* % of the parcel) with S2	1 or 0 or NA
M2	At least 2 clusters with more than *NPixClS2*	1 or 0 or NA
M3	DistNDVI > *ThrdNDVIdist*	1 or 0 or NA
M4	Compact S2 < *ThrdCompactS2*	1 or 0 or NA
Marker S1	**Description**	**Possible Value**
M5	More than one big cluster (>*PerHetero* % of the parcel) with S1	1 or 0 or NA
M6	At least 2 clusters with more than *NPixClS1*	1 or 0 or NA
M7	Compact S1 < *ThrdCompactS1*	1 or 0 or NA

The S2 markers are computed using the output of the extraction at the parcel level (see section 4.1.2), which is done as follows:

- **M1** is just another name for **Hete** output from S2. The marker is set as 1 when the maximum of the clP_{c} is above *PerHetero* and 0 otherwise.
- **M2** is set as 1 when the number of clusters with more than *NPixClS2* pixels is bigger than 2 and if M1 = 1. It's done using the **cl_{c}** outputs from S2. The value is 0 otherwise.
- **M3** is set as 1 when the **distNDVI** (output from S2) is above the *ThrNDVIdist* and 0 otherwise. The value of M3 can only be 1 if M1 is 1 since the distNDVI is calculated only when M1 = 1.

- **M4** is set as 1 when the **CompactA** output from S2 is above the *ThrdCompactS2* and 0 otherwise. The value of M4 can only be 1 if M1 is 1 since the Compact is calculated only when M1 = 1.

The S1 markers are computed using the output of the extraction at the parcel level (see section 4.1.3), which is done as follows:

- **M5** is just another name for **Hete** output of S1. The marker is set as 1 when the maximum of the **clP_{c}** is above *PerHetero* and 0 otherwise.
- **M6** is set as 1 when number of clusters with more than *NPixClS1* pixels is bigger than 2 and if M5 = 1. It's done using the **cl_{c}** outputs from S1. The value is 0 otherwise.
- **M7** is set as 1 when the **Compact** output from S1 is above the *ThrdCompactS1* and 0 otherwise. The value of M7 can only be 1 if M5 is 1 since the Compact is calculated only when M1 = 1.

Algorithm 4-4. Heterogeneity markers generation for S2

```
   if sum(v_count>NPixClS2) >= 2:
       df_cl1['M2'] = 1
   else:
       df_cl1['M2'] = 0

df_cl.loc[df_cl['Hete'].isin((0,3)),'M1'] = 0
df_cl.loc[df_cl['Hete'] == 1,'M1'] = 1
df_cl.loc[df_cl['distNDVI'] > ThrdNDVIdist,'M3'] = 1
df_cl.loc[df_cl['CompactA'] > ThrdCompactS2,'M4'] = 1
```

Algorithm 4-5. Heterogeneity markers generation for S1

```
   if sum(v_count>NPixClS1) >= 2:
       df_cl1['M6'] = 1
   else:
       df_cl1['M6'] = 0

df_cl.loc[df_cl['Hete'].isin((0,3)),'M5'] = 0
df_cl.loc[df_cl['Hete'] == 1,'M5'] = 1
df_cl.loc[df_cl['CompactA'] > ThrdCompactS1,'M7'] = 1
```

4.2.3 Outputs of the Parcel-level extraction and markers

For each period p, an output is generated for the S2 analys and:

- is a '.csv' named L4D_HeteS2_{site}_{year}_{p}.csv
- contains the same number of rows as the number of parcels in the declaration dataset.
- contains the columns described in Table 4-5

Table 4-5. Columns of the parcel-level extraction and markers results for the period p in the S2 analysis

Column name	Role	[format]
NewID	ID of the parcel in the Sen4CAP system	[int]
Hete	First heterogeneity flag. It can take two values. **0** if the parcel includes one cluster made of more than 90% of the pixels and to **1** otherwise.	0 or 1 [int]
clP_{c}	the number of pixels in the cluster (*c*)	[int]
cl_{c}	the percentage of pixels in the cluster (*c*) with regard to the total number of pixels within the parcel (*i*)	[float]
distNDVI	maximum distance of NDVI between 2 clusters	[float]

Compact	the averaged connectivity at the parcel-level	[float]
CompactA	the averaged connectivity at the parcel-level normalized by the surface of the parcel	[float]
M1	Value of the marker 1	[int]
M2	Value of the marker 2	[int]
M3	Value of the marker 3	[int]
M4	Value of the marker 4	[int]
HoleS2	Number of images in the period p for the parcel that is fully covered by NA (maximum is n_im_p);	[int]
HoleS2Part	Number of images in the period p for the parcel that is partially covered by NA (maximum is n_im_p).	[int]

For each period p, an output is generated for the S1 analysis and:

- is a '.csv' named L4D_HeteS1_{site}_{year}_{p}.csv
- contains the same number of rows as the number of parcels in the declaration dataset.
- contains the columns described in Table 4-6

Table 4-6. Columns of the parcel-level extraction and markers results for the period p in the S1 analysis

Column name	Role	[format]
NewID	ID of the parcel in the Sen4CAP system	[int]
Hete	First heterogeneity flag. It can take two values: **0** if the parcel includes one cluster made of more than 90% of the pixels, and **1** otherwise	0 or 1 [int]
clP_{c}	the number of pixels in the cluster (*c*)	[int]
cl_{c}	the percentage of pixels in the cluster (*c*) with regard to the total number of pixels within the parcel (*i*)	[float]
Compact	the averaged connectivity at the parcel-level	[float]
CompactA	the averaged connectivity at the parcel-level normalized by the surface of the parcel	[float]
M5	Value of the marker 5	[int]
M6	Value of the marker 6	[int]
M7	Value of the marker 7	[int]

4.3 Heteregeneity decision rules

The analysis of the period for decision on the heterogeneity should be as understandable as possible. We **strongly recommend to code this part as a Jupyther notebook**.

Decision about the heterogeneity is taken over a longer period PM (corresponding to at least 3 periods P), as explained in Figure 1-2.

The idea is to loop over each period p and analyse the markers in this p period and in the PM-1 periods after it. According to the value of the sum of all the markers in this PM period, a confidence level is given for the p period.

4.3.1 Period analysis for decision

Algorithm 4-6 requires inputs to produce a confidence level for heterogeneity detection based on an analysis conducted during each PM period. These inputs are described in Table 4-7

Table 4-7. Inputs of the period analysis for the decision algorithm

Inputs	Role	Default value [format]
site	Site of the sen4cap system	[string]
year	Year related to the season	YYYY [int]
decl_{site}_{year}	Parcel declaration after the preparation by the Sen4CAP system	[.csv]
d	the number of periods for the analysis at the PM period	[int]
L4D_HeteS2_{site}_{year}_{p}	Results of the Parcel extraction and markers generation for S2 for the all the period p of the monitoring period.	[.csv]

All the results from the parcel extraction and markers generation are concatenated into a single table called *dt_all* containing only the columns NewID, M1 to M5, HoleS2 and HoleS2Part and the period number.

l_marker is the number of markers (i.e. 4 when using S2). A loop over each parcel (i) is performed with the following steps:

1. Retrieving the markers in the dt_all of the parcel i and store it into dt_i
2. The HoleS2_All and the HoleS2_AllPart are respectively the sum of the HoleS2 for all the results and the sum of the HoleS2Part for all the results.
3. A loop over each PM is then performed:
    a. The pm is the period from p to $p + d$
    b. The markers from this pm period are selected and added together in the *sum_markers*. And the *M_sum* is a list containing the sum of each marker separately.
    c. The C_INDEX can take the following values and the PM start value is set:
        i. STRONG if the sum_markers is equal to l_marker*d (i.e. all the markers are true for the PM period)
        ii. MODERATE if the minium of M_sum is greater than d-2 and maximum of M_sum is d (i.e. one marker at least is verified for the pm period and the other markers should be verify at least 1 time (when d=3) in the pm period)
        iii. WEAK if only one of the markers is 0 and and maximum of M_sum is d (given here by the formula len([1 for x in M_sum if x > 0]) >= l_marker-1)
        iv. POOR if the sum_markers is greater than 2*d (i.e. at least half of the markers are 1).
        v. NA in every other situation.

    The C_INDEX will keep its maximum value until it is improved.

The output also contains the LC and the ShapeInd of the parcel analysed.

Algorithm 4-6. Period analysis for decision on the heterogeneity using only S2 results at the parcel

```
dt_i = dt_all.loc[dt_all['NewID']==i]
C_INDEX = 'NO'
P_Hete_L = np.nan
M1 = np.nan
M2 = np.nan
M3 = np.nan
M4 = np.nan
HoleS2_All = sum(dt_i['HoleS2'])
HoleS2_AllPart = sum(dt_i['HoleS2Part'])
#print(i)

pm_period_start = min(period_l)
pm_period_end = max(f['period'])-d

for p in range(pm_period_start,pm_period_end+1):
    pm = range(p,p+d)
    dt_i_m = dt_i.loc[dt_i['period'].isin(pm),markerL]
    sum_markers = dt_i_m.values.sum()
    M_sum = [sum(dt_i_m.iloc[:,0]),sum(dt_i_m.iloc[:,1]),sum(dt_i_m.iloc[:,2]),sum(dt_i_m.iloc[:,3])]
    if sum_markers == l_marker*d :
        #print('strong')
        #print(i)
        C_INDEX = 'STRONG'
        P_Hete_L = p
        M1 = M_sum[0]
        M2 = M_sum[1]
        M3 = M_sum[2]
        M4 = M_sum[3]

    elif (min(M_sum)>=(d-2)) & (max(M_sum)==d) & (C_INDEX!= 'STRONG'):
        C_INDEX = 'MODERATE'
        P_Hete_L = p
        M1 = M_sum[0]
        M2 = M_sum[1]
        M3 = M_sum[2]
        M4 = M_sum[3]

     elif (len([1 for x in M_sum if x > 0]) >= l_marker-1) & (max(M_sum)==d) & (C_INDEX not in
('STRONG','MODERATE')):
        C_INDEX = 'WEAK'
        P_Hete_L = p
        M1 = M_sum[0]
        M2 = M_sum[1]
        M3 = M_sum[2]
        M4 = M_sum[3]

    elif ((sum_markers >= 2*d) & (C_INDEX not in ('STRONG','MODERATE','WEAK'))):
        #print(f'markers sum : {dt_i_m.values.sum()} and c_index previous : {c_ind}')
        C_INDEX = 'POOR'
        P_Hete_L = p
        M1 = M_sum[0]
        M2 = M_sum[1]
        M3 = M_sum[2]
        M4 = M_sum[3]

dt_out = {
    'NewID' : i,
    'LC': lpis_csv.lc[i],
    'ShapeInd': lpis_csv.ShapeInd[i],
    'M1' : M1,
    'M2' : M2,
    'M3' : M3,
    'M4' : M4,
    'P_Hete_L' : P_Hete_L,
    'C_INDEX' : C_INDEX,
    'HoleS2' : HoleS2_All,
    'HoleS2_PM' : HoleS2_AllPart}
```

4.3.2 Output of the decision rules

The final result is a .csv table that gives an indication on the heterogeneity at the parcel level and on when this heterogeneity might have happened. Two others variables (HoleS2_All and HoleS2_AllPart) inform on the quality of the data used in the cluster and so might have an impact on the quality of the final decision.

At this stage, the decision is only based on S2.

This table is called **L4D_HeteDecisionS2_{site}_{year}.csv** and contains the columns described in Table 4-8

Table 4-8. Columns of period analysis for decision making.

Column name	Role	[format]
NewID	ID of the parcel in the Sen4CAP system	[int]
LC	Land Cover of the parcel as described in Table 2-4	[int]
ShapeInd	Shape index of the parcel	[float]
M1	Sum of the marker 1 for the *d* periods after the P_Hete_L (i.e. Sum of M1 used for the C_INDEX computation)	[int]
M2	Sum of the marker 2 for the *d* periods after the P_Hete_L (i.e. Sum of M2 used for the C_INDEX computation)	[int]
M3	Sum of the marker 3 for the *d* periods after the P_Hete_L (i.e. Sum of M3 used for the C_INDEX computation)	[int]
M4	Sum of the marker 4 for the *d* periods after the P_Hete_L (i.e. Sum of M4 used for the C_INDEX computation)	[int]
P_Hete_L	Period p where the maximum C_INDEX is observed	[int]
C_INDEX	Confidence level in the Heterogeneity detection for the period PM (d periods after the period p). It can take 5 values: 'STRONG', 'MODERATE', 'WEAK', 'POOR' and 'NA'	[string]
HoleS2_All	Number of missing data on the full parcel for the monitoring period	[int]
HoleS2_AllPart	Number of missing data on the part of the parcel for the monitoring period	[int]

5. Output

This section distinguishes 3 kinds of outputs, coming from the clustering component (section 3), from the statistics extraction and markers definition at the parcel-level (sections 4.1 and 4.2) and from the decision rules (section 4.3).

5.1 Output of the clustering

For period *P* and for each S2 or S1 tile (S1 tile corresponding to S1 data clipped over the S2 tile extent), four products are generated:

1) Raster of clusters without isolated pixels from the S2 clustering named ***L4D_ClS2_{site}_{ti}_{p}.tif***;
2) Raster of cluster without isolated pixels from the of S1 clustering named ***L4D_ClS1_{site}_{ti}_{p}.tif***;
3) Raster of connectivity (considered as a proxy of the compactness of the cluster analysed) without isolated pixels from the S2 clustering named ***L4D_ClS2_{site}_{ti}_{p}_Connect.tif***;
4) Raster of connectivity (considered as a proxy of the compactness of the cluster analysed) without isolated pixels from the S1 clustering named ***L4D_ClS1_{site}_{ti}_{p}_Connect.tif***.

5.2 Output of the parcel level extraction

For each period P, an output gives the results of the extraction and the marker generation at the parcel level based on the clustering outputs. This is done independently for S1 and S2.

- The **L4D_HeteS2_{site}_{year}_{p}.csv** table contains the columns as described in Table 4-5
- The **L4D_HeteS1_{site}_{year}_{p}.csv** table contains the columns as described in Table 4-6

5.3 Output of the decision rules

The final output of this processor is another .csv table that gives the confidence level on the detection together with the markers agreggated for the decision making.

This table is called **L4D_HeteDecisionS2_{site}_{year}.csv** and contains the columns described in Table 4-8.