

Correcting topological errors using SSNbler and QGIS

Erin E Peterson and Alan Pearse

May 6, 2024

Contents

1.	Introduction	1
2.	Example data and software requirements	2
3.	The Landscape Network.....	2
3.1	Valid node categories	2
3.2	Types of topology errors.....	3
3.3	Additional topological restrictions	4
4.	Identifying topological errors and restrictions in R.....	5
5.	Verifying and correcting topological errors in QGIS.....	7
5.1	Set up QGIS.....	8
5.2	Check the nodecat values.....	8
5.3	Edit complex confluences and downstream divergences manually.....	10
5.4	Remove true topological errors using v.clean	18
5.5	Correct the remaining topological errors.....	25
6.	References.....	31

1. Introduction

To generate a valid landscape network (LSN) with the SSNbler R package, users must ensure their digitised streams are free of topological errors and topological restrictions not permitted in SSN2. Vector streams data are more prone to topological errors, but they can also occur in streams data generated from digital elevation models. Depending on the extent of the study area and size of the stream network(s), the number of errors may be so large that it would be impractical to painstakingly correct them by hand. The aim of this tutorial is to demonstrate a semi-automated approach for removing topological errors in an LSN. It has been written for SSNbler v1.0.0 and assumes the user has some familiarity with QGIS \geq v3.24-5 and a good working knowledge of R Statistical Software version \geq 4.0.0 (R Core Team 2024) software.

2. Example data and software requirements

This tutorial comes with an example dataset and R script. The dataset can be found in the zip archive, **topology.zip**. The topology dataset should contain “topo_streams.shp” (and associated files), which is a shapefile of streams data containing two sub-networks. It should also contain the subfolder, “final_outputs,” which contains a reference version of the final error-free LSN created in this tutorial. The R script, CheckTopologyQGIS.R, contains the R code described in this tutorial.

For this tutorial, it is assumed that the topology dataset has been unzipped and stored in the folder **C:/temp/topology/**, and that a new folder called **C:/temp/topology/work** has been created for writing intermediate outputs to local files. The folder structure on the user’s computer should look something like this:

Name	Date modified	Type
final_output	6/05/2024 4:46 PM	File folder
work	16/04/2024 7:37 PM	File folder
topo_streams.cpg	6/05/2024 4:37 PM	CPG File
topo_streams.dbf	6/05/2024 4:39 PM	DBF File
topo_streams.prj	6/05/2024 4:37 PM	PRJ File
topo_streams.shp	6/05/2024 4:39 PM	SHP File
topo_streams.shx	6/05/2024 4:39 PM	SHX File

3. The Landscape Network

It is important to have a basic understanding of the structure of a Landscape Network (LSN) and the relationship between the edges and nodes within it. Streams data are represented as a collection of directed edges, where the directionality is determined by the digitized direction of the line features. A nodes dataset (nodes.gpkg) is created when the LSN is generated in R using the `lines_to_lsn` function in SSNbler. Each edge is associated with two nodes, which correspond to the upstream and downstream endpoints of the edge. When more than one edge flows into or out of the node, they *share* a node. Thus, there should always be a single node at the intersection of edges (Fig. 1). If there is more or less than one node at an intersection, it is a topological error. If these errors are not corrected, the connectivity between line features and the observed and prediction sites associated with them will not be accurately represented in the SSN object or the spatial statistical models subsequently fit to the data. Correcting these errors is often the most time-consuming step in the pre-processing and modelling workflow.

3.1 Valid node categories

There are four topologically valid node categories in a LSN:

- **Source:** Water flow originates at these nodes and flows downstream. An edge that begins at a source node is called a headwater segment. Source nodes do not receive flow from any upstream edges.

- **Outlet:** Flow terminates at these nodes. This node is the most downstream point in the edge network. We refer to an edge that terminates at an outlet as an outlet segment. Note that a streams dataset may contain multiple subnetworks with associated outlets.
- **Confluence:** Multiple edges converge, or flow into, a single node.
- **Pseudonode:** One edge flows in and one edge flows out of a single node. While these nodes are not topologically problematic, an excessive number of pseudonodes can slow down geoprocessing operations for large datasets.

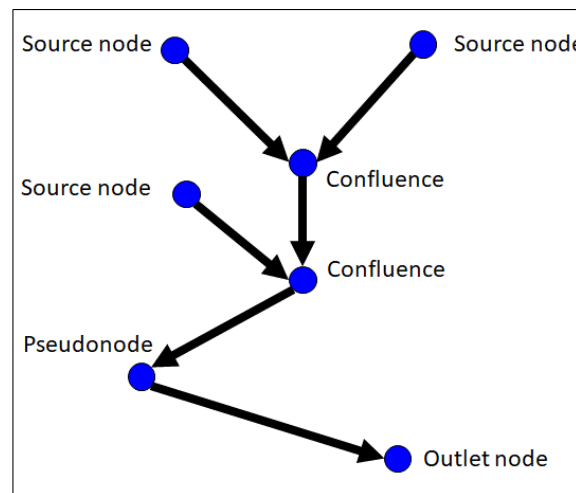


Fig. 1. An illustration of a landscape network (LSN). Nodes are denoted by blue circles, with the node category labelled. Edges are denoted by black arrows, with the arrow indicating flow direction (i.e. digitized direction).

3.2 Types of topology errors

When topological errors exist in an LSN, the flow connectivity is misrepresented, which can negatively affect statistical model outputs. Therefore, these errors must be corrected prior to fitting statistical models. Three common topological errors are recognised (Fig. 3):

1. **Unsnapped node:** occurs when two edges should be connected by a common node but are instead separated by a small distance (Fig. 2a).
2. **Dangling node:** occurs when the end node of an edge is close to another edge that it should be connected to but does not touch.
3. **Intersection without nodes:** two edges in a stream network should never intersect without a node present. This error occurs when one edge feature crosses another without a node at the intersection (Fig. 2c) or the end node of one edge feature touches another edge at any point other than the end nodes (Fig. 2d). Note that the 'outlet node' at the end of the offending line segment may also be considered a dangling node (Fig. 2c, yellow dot).

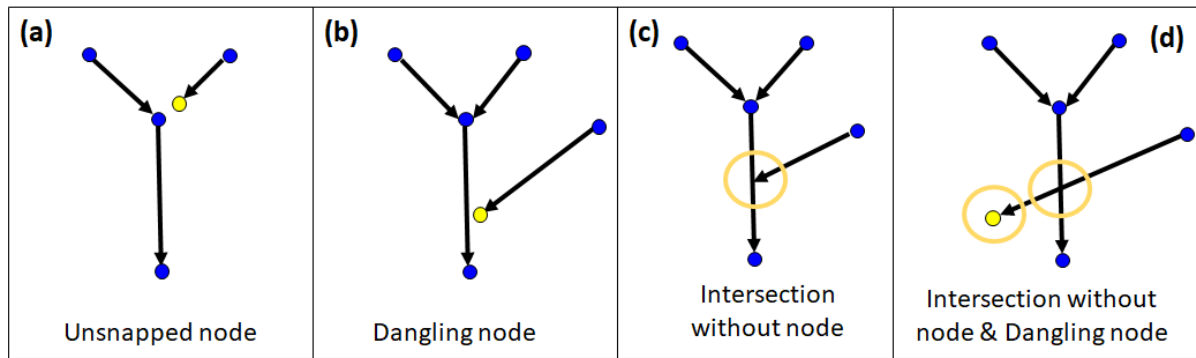


Fig. 3. Illustrations of topological errors in a stream network. The nodes represented as blue dots, with the node of interest shown as yellow dots and circles. The edges are represented using black arrows pointing in the direction of flow. Three types of topological errors are shown: a) Unsnapped node, b) Dangling node, c) Intersection without a node, and d) Intersection without node and Dangling node.

3.3 Additional topological restrictions

In addition to true topological errors, some additional topological restrictions are imposed on the nodes (Fig. 3). These include:

- **Converging node:** occurs when two edges flow into a node and no edges flow out (Fig. 3a).
- **Complex confluence:** occurs when more than two edges flow into a node (Fig. 3b). No more than two edges can flow into a confluence. If this condition is violated, the node is called a *complex confluence* (Fig. 2c).
- **Downstream divergence:** occurs when more than one edge flows out of a node (Fig. 3c). This topological condition occurs naturally in braided streams (Fig. 3d), but is not allowed in a LSN.

Although these topological conditions appear naturally in stream networks, they complicate the derivation of flow-based dependence structures in LSNs. Therefore, these cases must be adjusted or corrected.

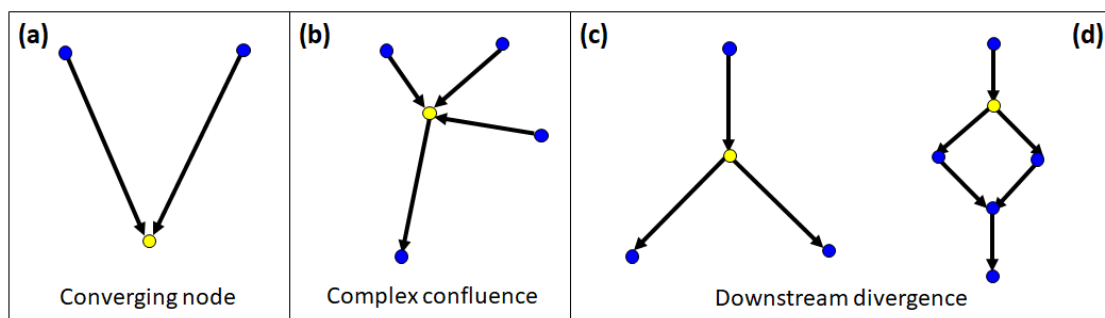


Fig. 2. Illustrations of topological features in stream networks that are restricted in landscape stream networks. Nodes are drawn as circles, and the edges are drawn as arrows pointing in the direction of flow. In each panel, the problem node is highlighted in yellow; the associated label identifies the topological restriction that is violated by the node.

The purpose of this tutorial is to demonstrate a geoprocessing workflow that may help users eliminate topological errors and restricted conditions from their stream network data using QGIS. Some steps in this tutorial are semi-automated and take advantage of existing processing tools in QGIS. Other steps require manual adjustments to be made to line segments. In general, error-correction is an iterative process that may involve repetition of manual and semi-automatic steps in various combinations, and this is showcased in this tutorial.

4. Identifying topological errors and restrictions in R

Topological errors (Fig. 2) and restrictions (Fig. 3) can be identified using the `lines_to_lsn` function in SSNbler. This function does three main things:

1. Builds the LSN;
2. Checks network topology and fixes minor topological errors (optional); and
3. Writes output files to a local directory.

We will begin by walking through these steps.

Open `CheckTopology.R` in RStudio or any other IDE for R. Load the relevant R packages and set your working directory, which may be different to the one below:

```
## Load SSNbler and other useful packages
library(SSNbler)
library(sf)
library(dplyr)
library(purrr)

## Set working directory
setwd("C:/temp/topology")
```

Next create an `sf` object containing the streams data and build the initial LSN using `lines_to_lsn`. In this case, we are importing a shapefile, but there are no constraints on the data format, except that it must have `LINESTRING` geometry.

Checking topology using `lines_to_lsn` is optional and there are a few arguments that must be set to do so:

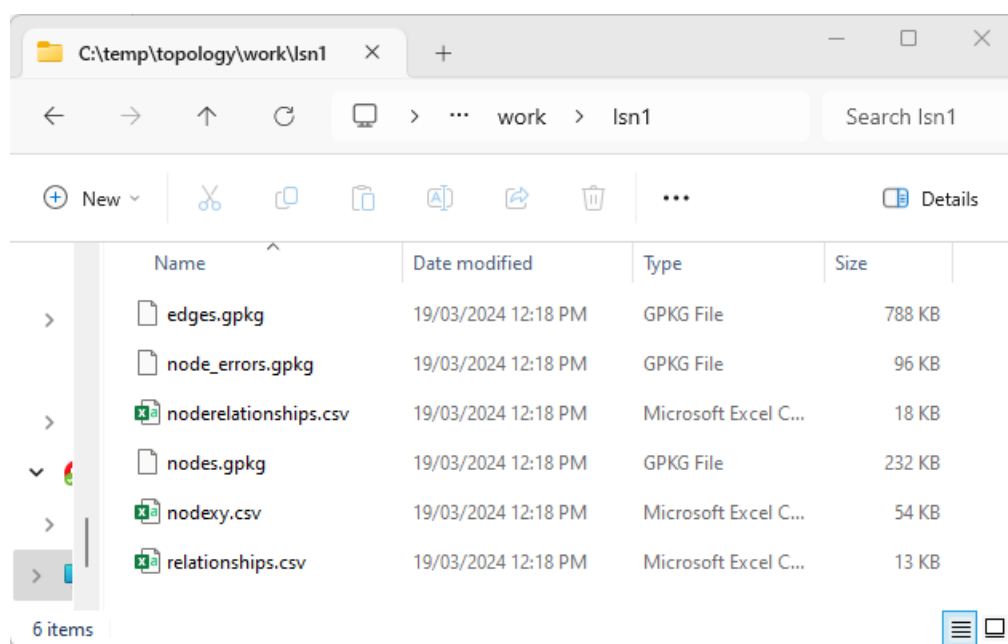
- **check_topology**: Logical indicating whether the topology should be checked;
- **snap_tolerance**: Distance in map units. Two nodes separated by a distance less than or equal to `snap_tolerance` are assumed to be connected.
- **topo_tolerance**: Distance in map units. Two nodes separated by a distance less than or equal to `topo_tolerance` are flagged as potential topological errors in the network.

```
## Import the streams dataset as an sf Object
river_net<- st_read("topo_streams.shp")

## Build the initial LSN and check the topology
lsn_path1<- "c:/temp/topology/work/lsn1"
edges<- lines_to_lsn(
  streams = river_net,
  lsn_path = lsn_path1,
  snap_tolerance = 1,
  check_topology = TRUE,
  topo_tolerance = 20,
  overwrite = TRUE,
  verbose = TRUE,
  remove_ZM = TRUE)
```

The arguments `snap_tolerance` and `topo_tolerance` are both provided in map units. It is generally a good idea to set the `snap_tolerance` to a relatively small value compared to the `topo_tolerance` argument. Nodes separated by a Euclidean distance $\leq \text{snap_tolerance}$ are assumed to be connected. If this distance $\leq (\text{snap_tolerance}/10000)$, the nodes are automatically snapped when `check_topology = TRUE`. When $(\text{snap_tolerance}/10000) \leq \text{distance} \leq \text{snap_tolerance}$, the nodes are not snapped and are instead flagged as potential errors for the user to check. Similarly, when $(\text{snap_tolerance}) < \text{distance} \leq \text{topo_tolerance}$, nodes are flagged as potential errors. To learn more about the other function arguments, simply type `help(lines_to_lsn)`.

The function `lines_to_lsn` always saves five output files to `lsn_path` when it runs successfully. These include edges and nodes in geopackage format (.gpkg) and three .csv files containing tables that describe the topological relationships in the LSN (`nodexy.csv`, `noderelationships.csv`, and `relationships.csv`). If potential topological errors are found in the network, there will be an additional file named `node_errors.gpkg`.



The contents of the new folder can also be checked from R:

```
> ## Check output files. If node_errors.gpkg exists, then there are
+ ## potential errors to check
+ list.files(lsn_path1)
[1] "edges.gpkg"          "node_errors.gpkg"      "noderelationships.csv"
[4] "nodes.gpkg"          "nodexy.csv"           "relationships.csv"
```

In this case, node_errors.gpkg exists. Run the following code to look at and summarize the types of topological errors and restrictions identified:

```
> ## Import node errors and format columns
+ node_errors <- st_read(paste0(lsn_path1, "/node_errors.gpkg"),
+                        quiet = TRUE) %>%
+   modify_if(is.character, as.factor)
+ ## Summarise
+ summary(node_errors)
```

pointid	nodecat	error
Min. : 275	Confluence:15	Complex Confluence : 1
1st Qu.:1325	Outlet :21	Converging Node : 5
Median :1337		Dangling Node :19
Mean :1162		Downstream Divergence : 1
3rd Qu.:1351		Intersection Without Node: 8
Max. :1371		Unsnapped Node : 2
NA's :11		

```

      geom
POINT      :36
epsg:5070  : 0
+proj=aea ...: 0
```

The object node_errors contains three columns in addition to the geom (or geometry) column that will be useful for checking topology errors.

- **pointid:** point identifier for each node.
- **nodecat:** node category, which can take on values of Source, Confluence, Pseudonode, and Outlet.
- **error:** node error type, which includes topological errors (Unsnapped Node, Dangling Node, and Intersection Without Node) and restrictions (Complex Confluence, Converging Node, and Diverging Node).

Note that the pointid and nodecat columns are also found in nodes.gpkg. It is also important to recognize that node_errors indicate *potential* topological errors. The user must verify whether these are true errors and then edit the topology accordingly. At this point, we move to QGIS where the topological editing will occur.

5. Verifying and correcting topological errors in QGIS

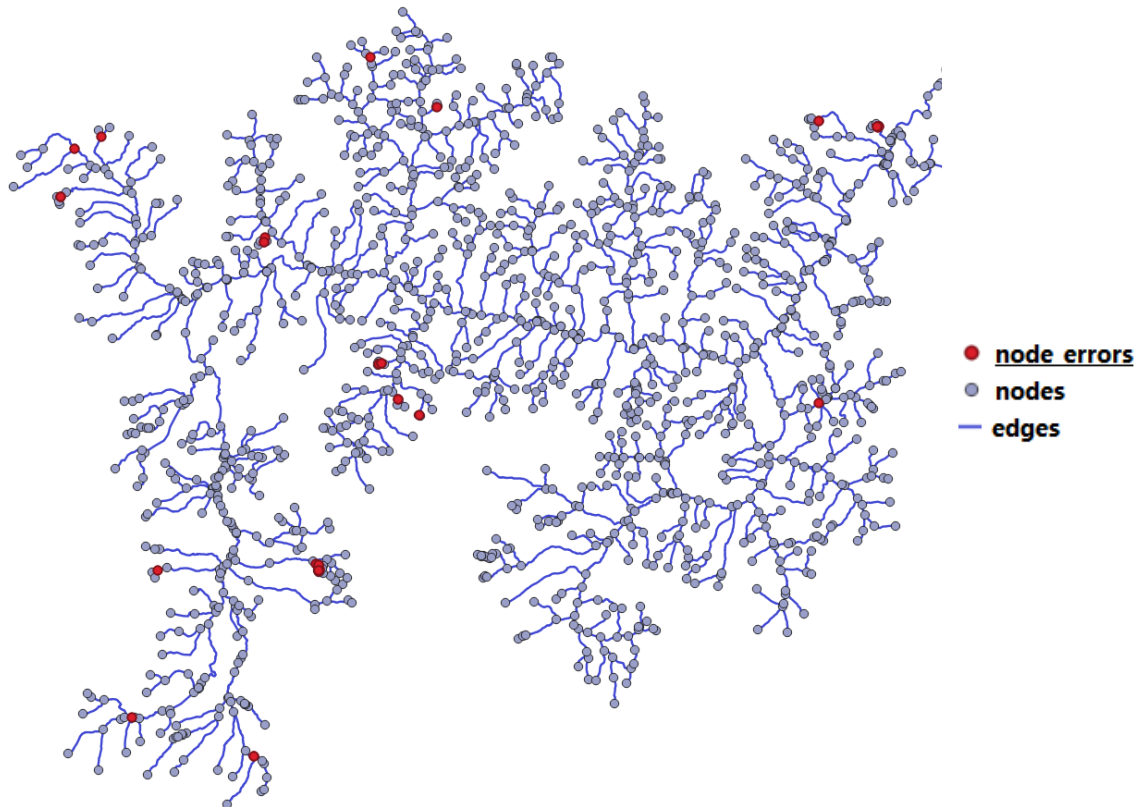
We assume that the reader has a working knowledge of QGIS and is familiar with QGIS editing tools. For those that need additional help, the QGIS documentation is a good place to start

(<https://docs.qgis.org/3.34/en/docs/index.html>) and Chapter 16 of the QGIS User Manual focuses on

working with, and editing, vector data. Online discussion forums are also a good option – remember, it's rare to have a question that has not be asked in some form before!

5.1 Set up QGIS

- a. Open QGIS and load edges.gpkg, nodes.gpkg, and node_errors.gpkg. Change the symbology of the layers to a colour scheme that makes it easy to distinguish between nodes and node_errors. Your data should look something like this:



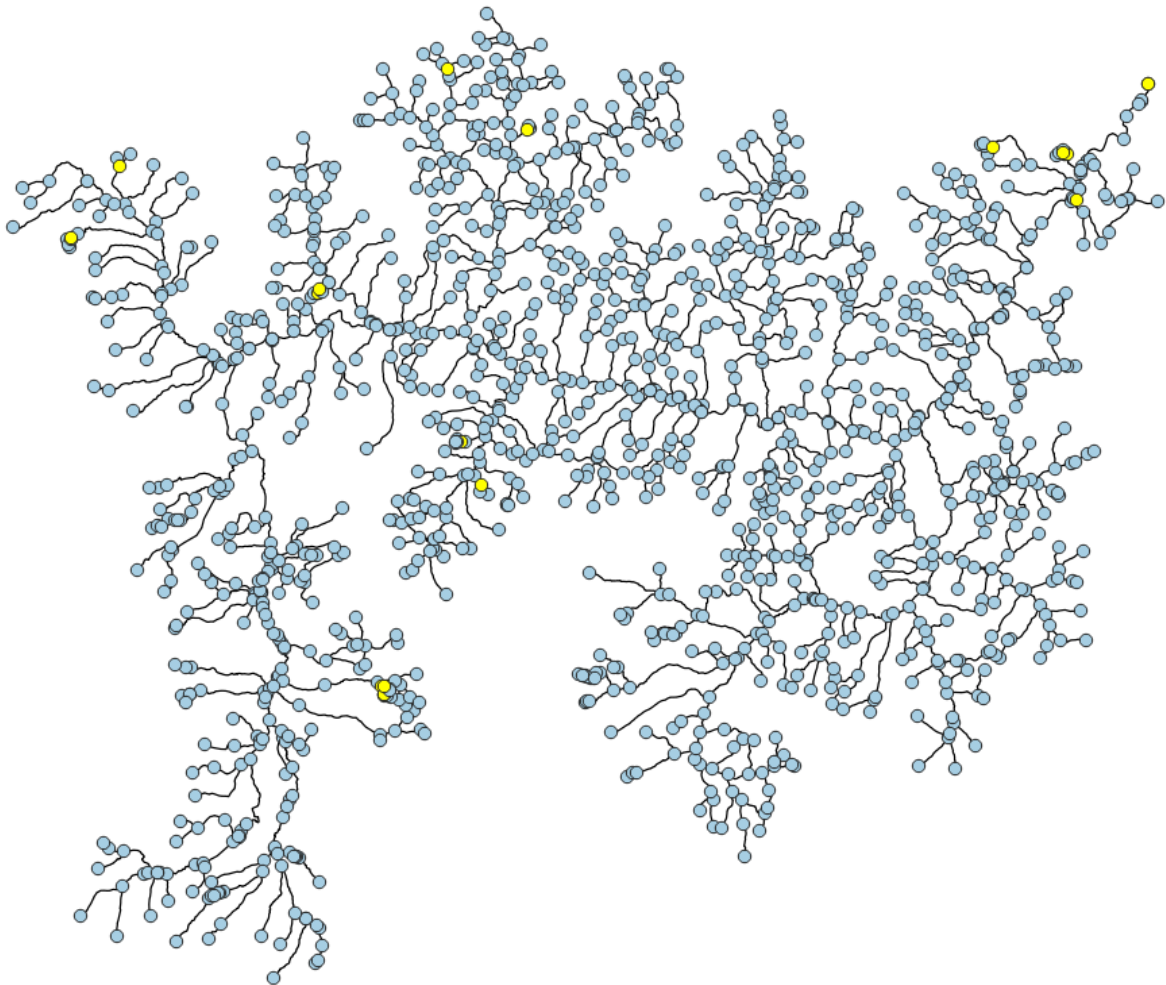
- i. Save the QGIS project. In the main menu, select Project, scroll down, and click on Save As, and save the Project to a local directory.

5.2 Check the nodecat values

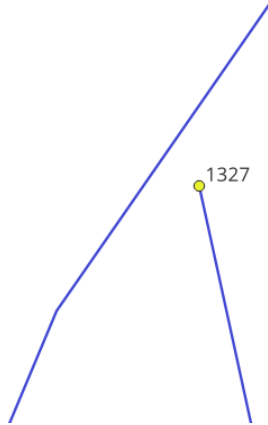
The first step should always be to check the nodes (not node_errors) on the network. In particular, the number and positions of Outlet nodes should be investigated to identify obvious errors. For example, having a large number of outlets may indicate the presence of topological errors, especially when they are found in the interior of a stream network. The extra, erroneous Outlets are usually the result of a failure to snap the end nodes of edges segments (e.g. Unsnapped Nodes or Dangling Nodes).

1. Open the nodes attribute table by right-clicking on it in the Layers panel, scrolling down, and clicking Open Attribute Table.
2. Select the rows of the attribute table where the “nodecat” = “Outlet”. There should be 24 Outlet nodes selected.

3. Close the attribute table and uncheck the node_errors in the Layer Panel. The Map window should look like this. The selected Outlet nodes are highlighted here in yellow, but the selection colour may vary depending on user default settings.



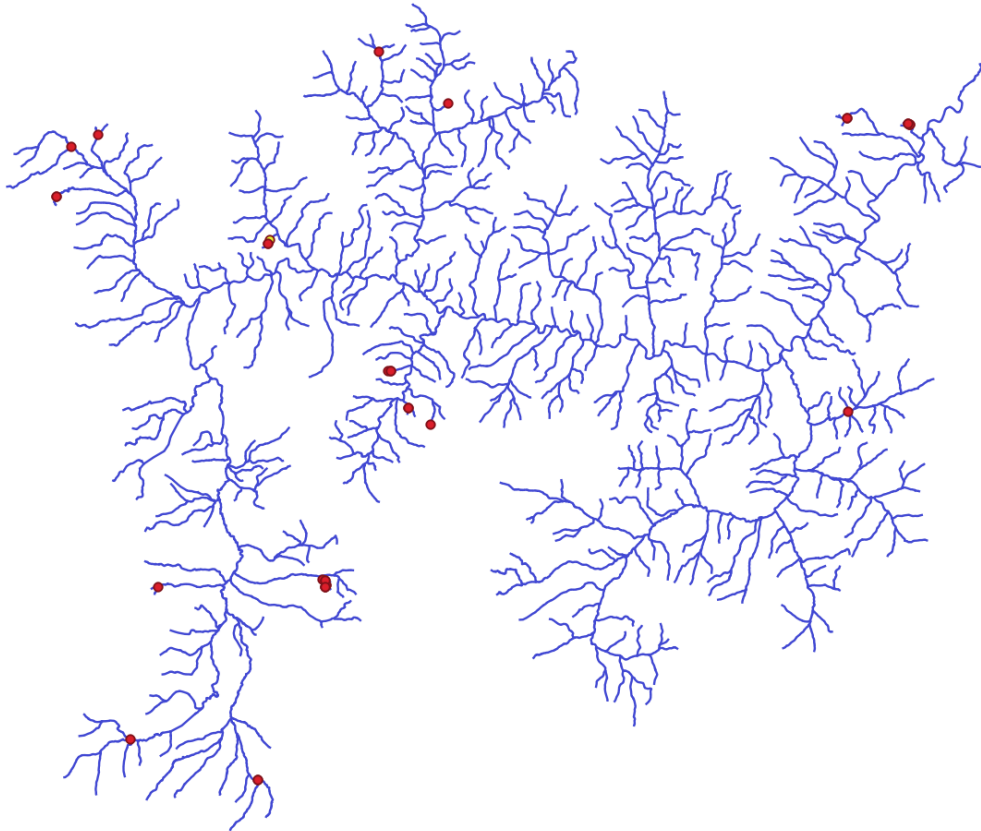
There are only 2 true outlets in this dataset, but 24 Outlet nodes have been identified. Notice the Outlets highlighted in the interior of this stream network. These are unlikely to be true outlets. Zooming in on one of the selected Outlets (pointid = 1327) reveals a situation like the one below, where a node has been identified as an outlet because it has not been snapped to the neighbouring stream segment.



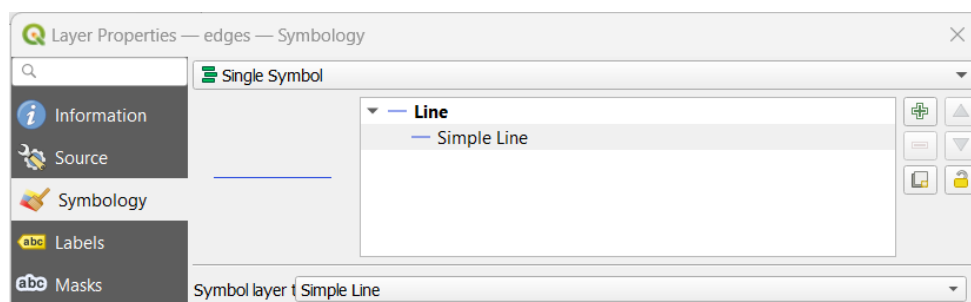
4. Investigate the Outlet nodes. For large datasets with many Outlets, manual inspection of every node will not be feasible. In this case, simply check a few at random, but pay special attention to any Outlets located in the interior of the network.
5. Remove nodes from the Layers Panel because it will not be needed again for this tutorial. However, keep in mind that the number and positions of Outlets should be checked every time an LSN is generated by lines_to_lsn.

5.3 Edit complex confluences and downstream divergences manually

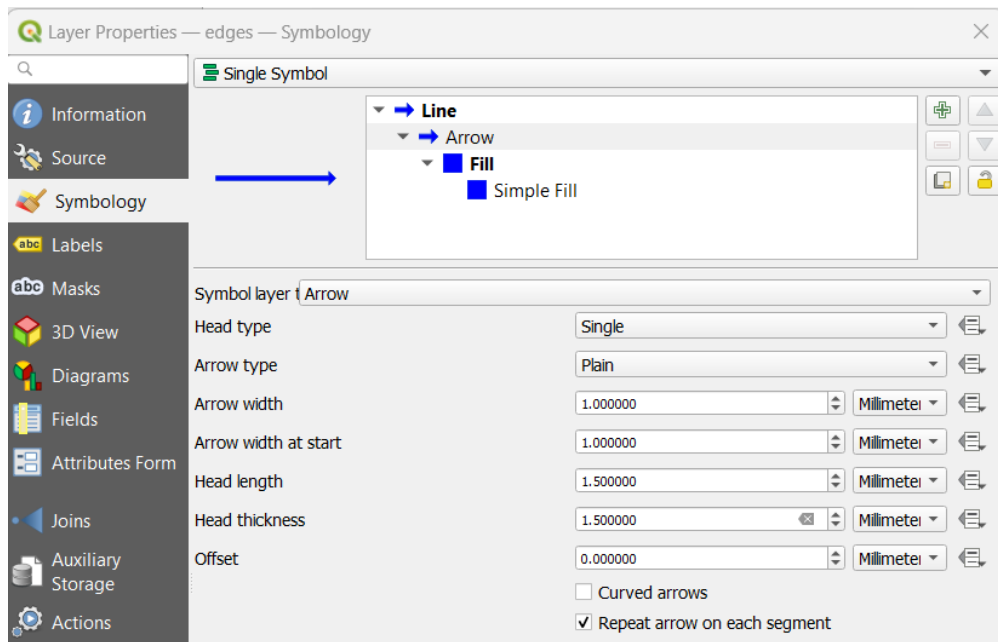
1. Turn on node_errors in the Layer Panel (if it is turned off). The Map view should look something like this, where points represent node_errors and lines represent edges.




2. Change the symbology of edges to lines capped with arrows indicating the digitized direction and direction of flow in the LSN.
 - a. Right-click on edges in the Layers panel, scroll down and select Properties (it should be down the bottom). This will open Layer Properties window where the symbology for edges can be changed.
 - b. Click on Symbology in the menu on the left and click on the text that reads “Simple Line” in the top menu. This will bring up additional options for the line symbol.



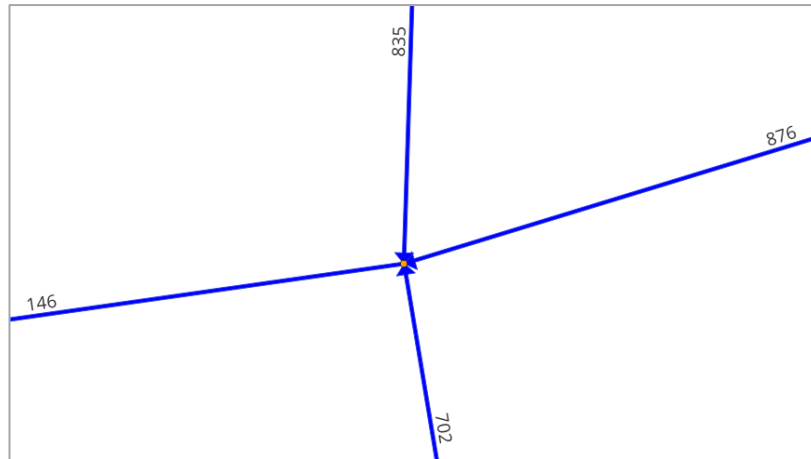
- c. Click on the Symbol layer dropdown menu and change Simple Line to Arrow. Deselect the Curved arrows box as shown below. Click Apply and then Ok to close the Layer Properties window. The result will look quite messy because QGIS places more than one arrow head on each line segment, but this will improve once we zoom in.



3. Open the attribute table for node_errors and select "error" = 'Complex Confluences'.
 - a. Right click on node_errors in the Layers Panel and click on Open Attribute Table.
 - b. The node_errors include one Complex Confluence. Select this record manually or using the Select by Expression tool, , and the expression:

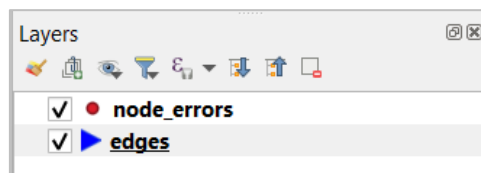
"error" = 'Complex Confluence'
 - c. Close the attribute table and zoom into the node_error highlighted in yellow. You may have to temporarily turn off the edges in the Layers panel to see it.

In the image below, the yellow point represents the complex confluence, and the numbers are the rid values for each edge. Notice there are three upstream edge features flowing into the node (rid = 702, 835, and 876) and one downstream edge flowing out (rid = 146). Complex confluences are not allowed and so one of the upstream edges must be edited. If one of the upstream edges is a small headwater stream, it may be appropriate to simply delete it. If not or there is another reason to retain all three edges, the end node of one upstream edge must be moved slightly up or downstream of the of the complex confluence. We describe how to do that next.

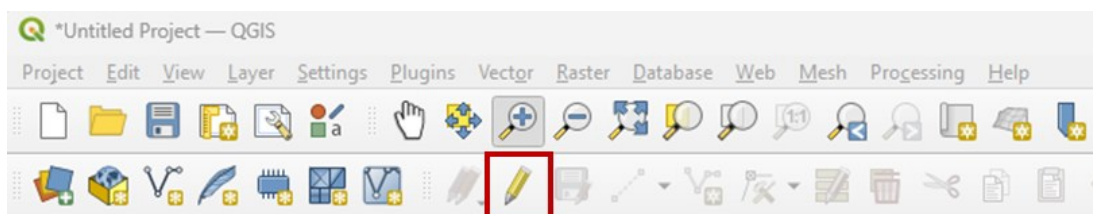


4. Remove the complex confluence by editing the edges

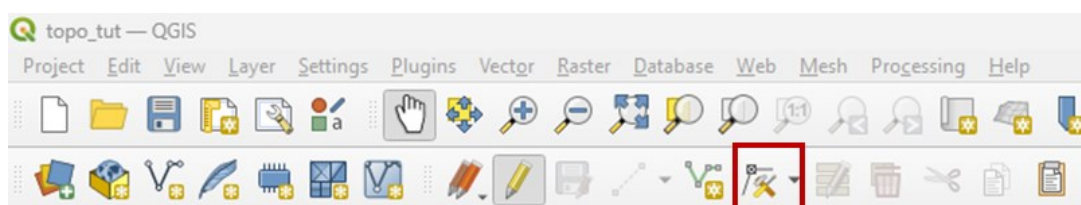
- d. Select edges in the Layers Panel by clicking on it.



- e. Turn on the layer editor by clicking the Toggle Editing button in the Digitizing toolbar (the button indicated below with a red square). If the Digitizing tool bar is not available, right click anywhere in the Menu, scroll down to Toolbars, and check the box next to the Digitizing toolbar to add it. The edges are now in editing mode.

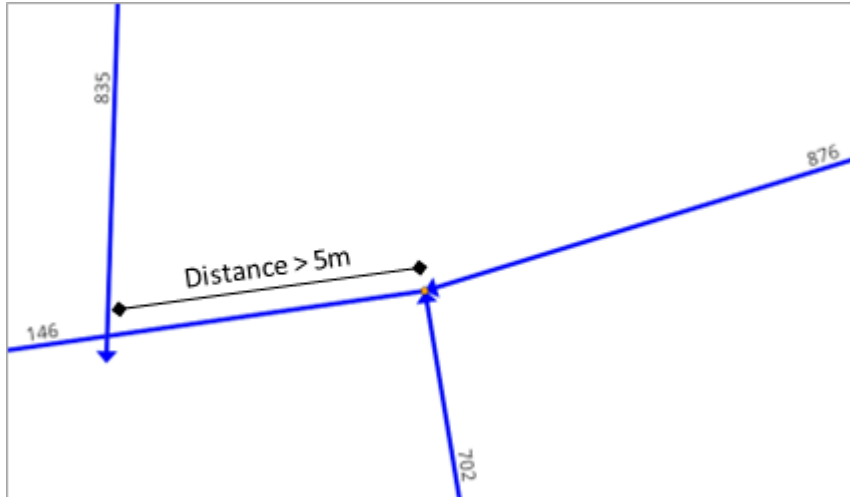



- f. Turn on the Vertex Tool (Current Layer), which is shown below in the red box. This opens the Vertex Editor panel.



- g. Hover over the upstream line feature with rid = 835 and click to select it. The line segment can now be moved slightly up or downstream of the node. Do not worry if the edited line feature intersects or does not end exactly at the end node of another

line feature. These new topological errors will be fixed in subsequent steps. However, the line segment must be moved a greater distance than both the snap_tolerance used in subsequent calls to lines_to_lsn (1m) and the snapping threshold we use later in the tutorial (5m). This helps to avoid snapping issues later. Do this for every complex confluence identified in node_errors.



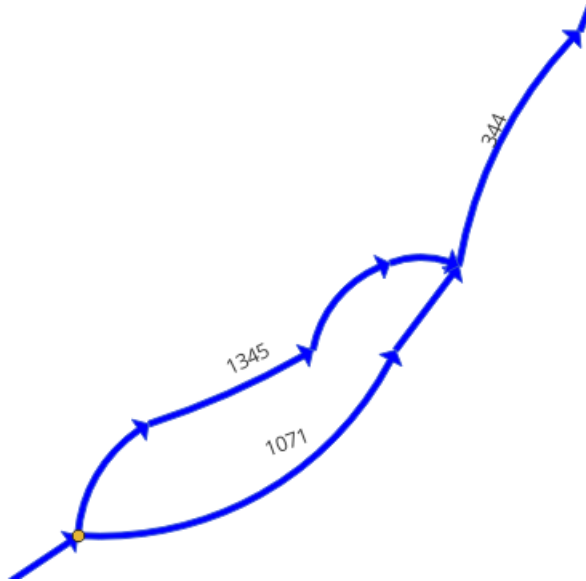
- h. After editing each complex confluence, save the edits to the current layer by clicking the “Save Layer Edits” button, , in the menu ribbon at the top.

5. Edit downstream divergences

- a. Open the node_errors attribute table and use the Select by Expression tool to select
“error” = ‘Downstream Divergence’

Zoom in to the selected point.

The yellow point in the image below is the Downstream Divergence and the numbers represent the edge rid values. You can see that one edge feature flows into the node and two features flow out (rid = 1345 and 1071), which is not permitted. One of these edges will need to be edited to remove the downstream divergence and there are two general approaches for doing that.



The easiest way to remove a downstream divergence is to identify which edge represents the main channel and simply delete the other edge (e.g. delete rid = 1345 or 1071). Alternatively, one of the edges can be disconnected from the node, which allows both edges to be retained in the LSN. The latter approach is described in the next steps.

- b. Ensure that edges is selected in the Layers Panel and then use the Select Features By



Area tool to select to select edge rid = 1345 and rid = 1071.

- c. Open the edges attribute table. In the bottom left corner, click the dropdown menu by Show All Features and change it to Show Selected Features. The two selected edge records should be highlighted.

edges — Features Total: 1345, Filtered: 2, Selected: 2

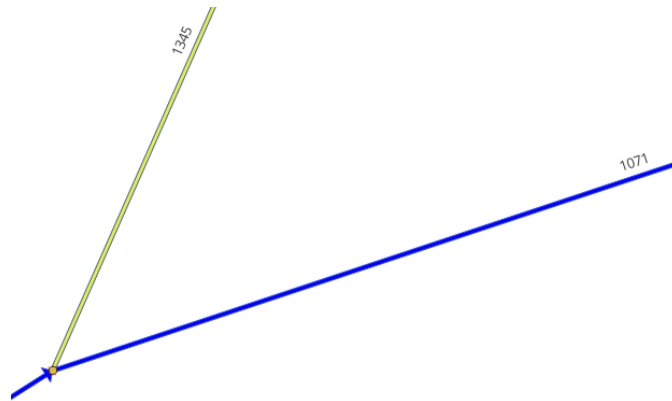
123 fid = 123 [Update Filtered] [Update Selected]

	rid	GNIS_NAME	StreamOrde	AreaSqKM	TotDASqKM	Length
1	1071	Salmon River	6	10.7721	4731.9048	205.493189
2	1345	Salmon River Trib	1	1e-05	17.4423	212.8323774

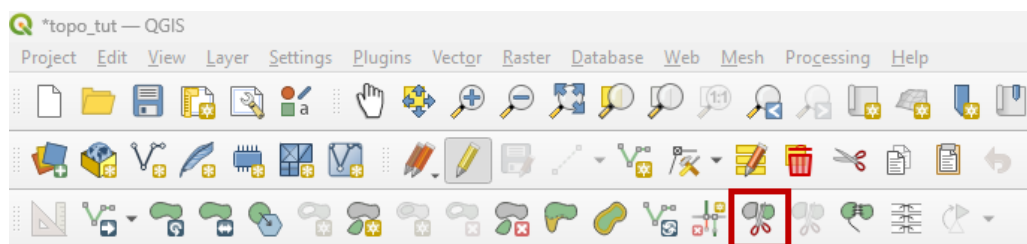
Show Selected Features

- d. Compare the attributes for the two edge values and choose which feature to edit. In this example, you can see that rid = 1071 has a much larger stream order (StreamOrde) and drainage area (AreaSqKM) than rid = 1345. In addition, the name (GNIS_NAME) suggests that 1345 is a tributary of 1071. Therefore, edits will be made to rid = 1345. In practice, the decision may be more subjective than this and you'll simply have to use your best judgment.

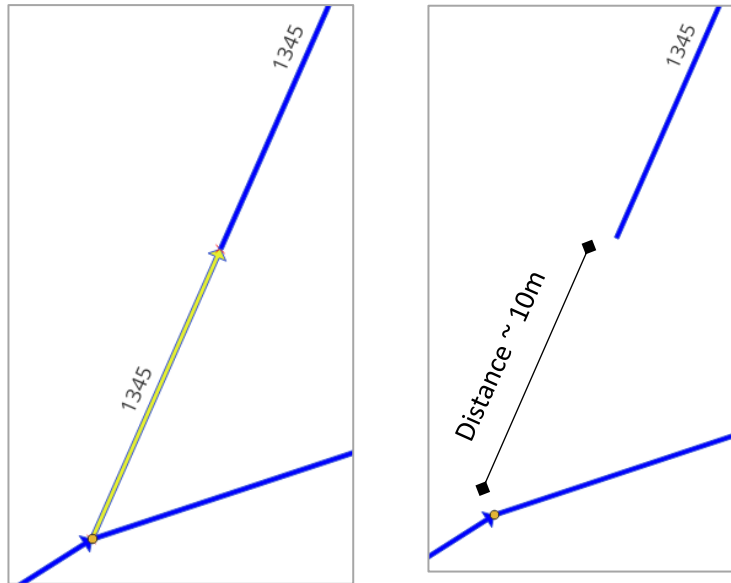
- e. Clear the selection, select the edge feature with rid = 1345, and zoom in to the downstream divergence.



- f. The next step is to split edge 1345 downstream of the downstream divergence using the Split Features tool in the Advanced Digitizing Toolbar shown in the red box below. If the tool is not visible, you can enable it by going to **View > Toolbars > Advanced Digitizing Toolbar**.



- g. Identify where on the line feature you would like to make the split. Make sure that the distance between the node and the split is greater than the any snap tolerance values used in past and future lines_to_Isn and QGIS editing step. In this example **split the line feature approximately 10m downstream of the Diverging node**.
- h. With the line feature selected, click on the Split Features tool and the pointer will turn into a cross hairs symbol. Hover near the selected edge and left click once to activate the tool. Move the cross hairs across the feature where you would like to make the split, then left click again. A red line will appear. Right click to make the split. Two line features should now be present, as shown in the left panel below.
- i. Select the edge feature closest to the downstream divergence node_error if it is not already selected (left panel below). Click on the Delete Selected tool, or simply press the delete button on your keyboard, to delete the feature. A dialog box may open asking you to confirm that you want to delete 1 feature from the layer edges. Click the Delete 1 Features(s) button. Zoom back out and see that the edge with rid = 1345 is still present, but it is no longer connected to the upstream node (right panel below).



- j. Click on the Save Layer Edits button to save these edits and then the Toggle Editing tool to stop editing the edges.
- k. Remove all layers from the Layers menu.

6. **Return to R** and run `lines_to_lsn` on the edited edges

- a. Load the SSNbler library and set the working directory.
- b. Import the edited edges and run `lines_to_lsn` with `check_topology = TRUE` to create a new LSN. Notice that this new LSN is saved to a new directory defined in `lsn_path2`.

```
## Import edited edges
edges1 <- st_read(paste0(lsn_path1, "/edges.gpkg"))

## Build the initial LSN and check the topology
lsn_path2<- "c:/temp/topology/work/lsn2"
edges<- lines_to_lsn(
  streams = edges1,
  lsn_path = lsn_path2,
  snap_tolerance = 1,
  check_topology = TRUE,
  topo_tolerance = 20,
  overwrite = TRUE,
  verbose = TRUE,
  remove_ZM = TRUE)
```

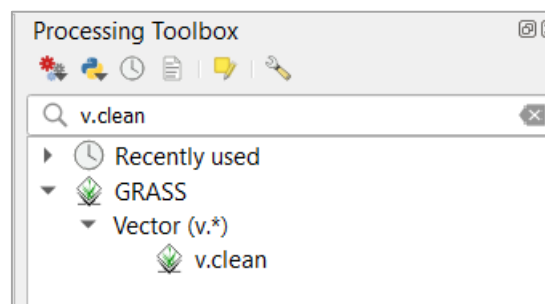
- c. Import `node_errors` and summarise as we did previously. There should be no errors labelled as Complex Confluence or Downstream Divergence. There may be new errors where the Complex Confluence and Downstream Divergences were previously located, but these will be fixed in subsequent steps.

5.4 Remove true topological errors using v.clean

1. **Return to QGIS** and add the updated edges.gpkg and node_errors.gpkg created in the last step from: C:/temp/topology/work/lsn2. Accept the default symbology for edges or change it based on your preferences.
2. Go to Processing > Toolbox in the top menu ribbon to open the Processing Toolbox, which should appear in a window on the right-hand side of the screen (by default).
3. Check that the GRASS toolbox is enabled and visible in the Processing Toolbox. If the GRASS toolbox is missing,
 - a. Click on the Plugin menu and select Manage and Install Plugins to open the Plugin window.
 - b. Click on Installed on the left and check the box next to GRASS GIS provider to enable it. Close the window.

The GRASS toolset should now be visible in the Processing toolbox. Note that GRASS is a Core plugin installed with QGIS by default and no separate GRASS installation or configuration should be required.

4. In the Processing Toolbox menu, search for the GRASS tool **v.clean**. Double-click on v.clean to open the tool.



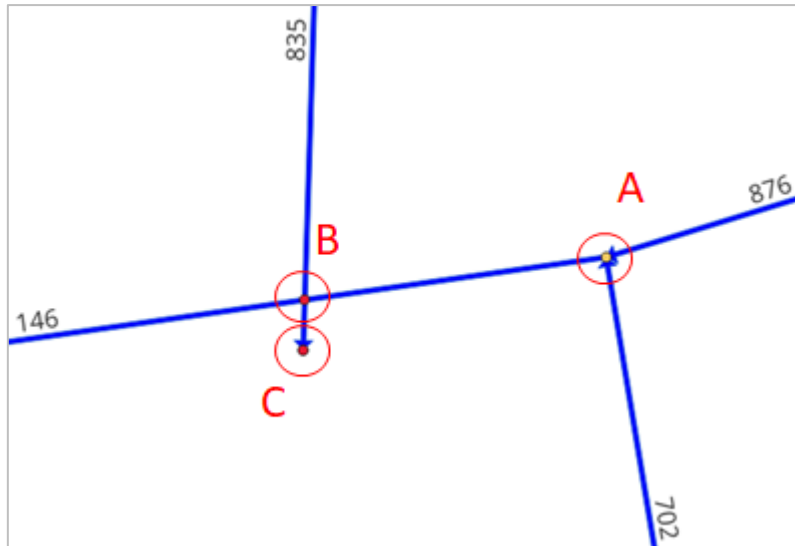
The v.clean tool will be run several times in the next steps as part of an iterative error correction process. To learn more about v.clean and the arguments, see the GRASS GIS manual (<https://grass.osgeo.org/grass83/manuals/v.clean.html>).

5. Iteratively run the v.clean tool four times to remove topology errors.

In this section we use the Converging Confluence node error that was corrected in Section 5.3 as an example.

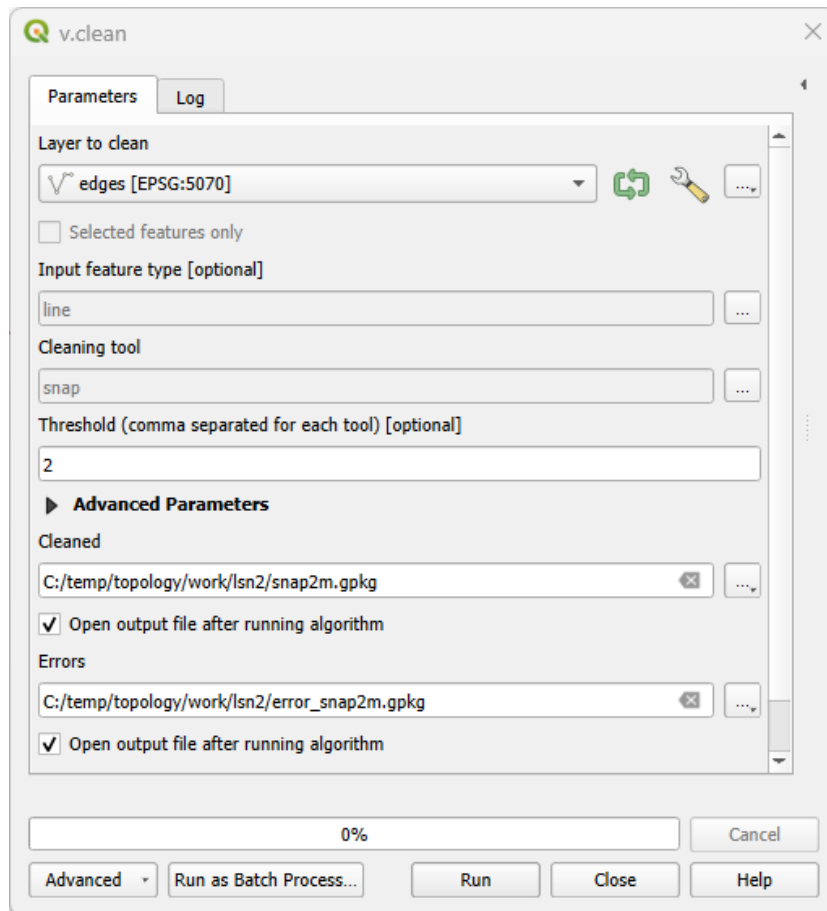
- a. In nodes, select and zoom to the node "pointid" = 275.

The View should show something like the image below (without the labels), where the edges are displayed as blue lines and the corrected Complex Confluence (A) as a yellow dot. Two node errors (red dots) were introduced when the Complex Confluence was removed: B) Intersection Without Node and C) Dangling Node.



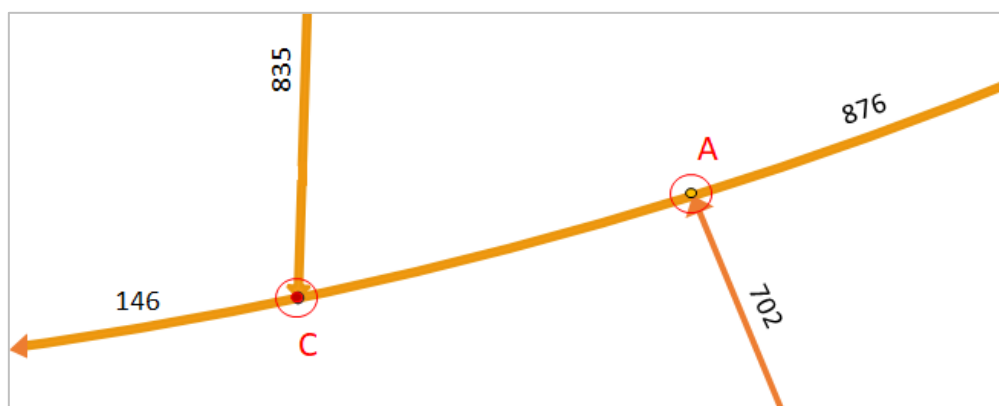
- b. **Run v.clean 1:** First, v.clean will be used to snap any unsnapped vertices over very small distances using the Cleaning tool 'snap'. **It is crucial that this step is undertaken first** to prevent snapping errors when a larger snapping distance is used in a future step.

Supply arguments to v.clean as shown below. In this example the snap_tolerance argument passed to lines_to_lsn was 1m and so we set the snapping threshold slightly higher at 2m. The output Cleaned dataset, snap2m, is an edited version of edges that has been snapped, while error_snap2m only contains the edge features that were edited. The two output datasets can be compared using the common 'cat' column to more easily identify where edits were made. Click the Run button and then Close once v.clean is finished.

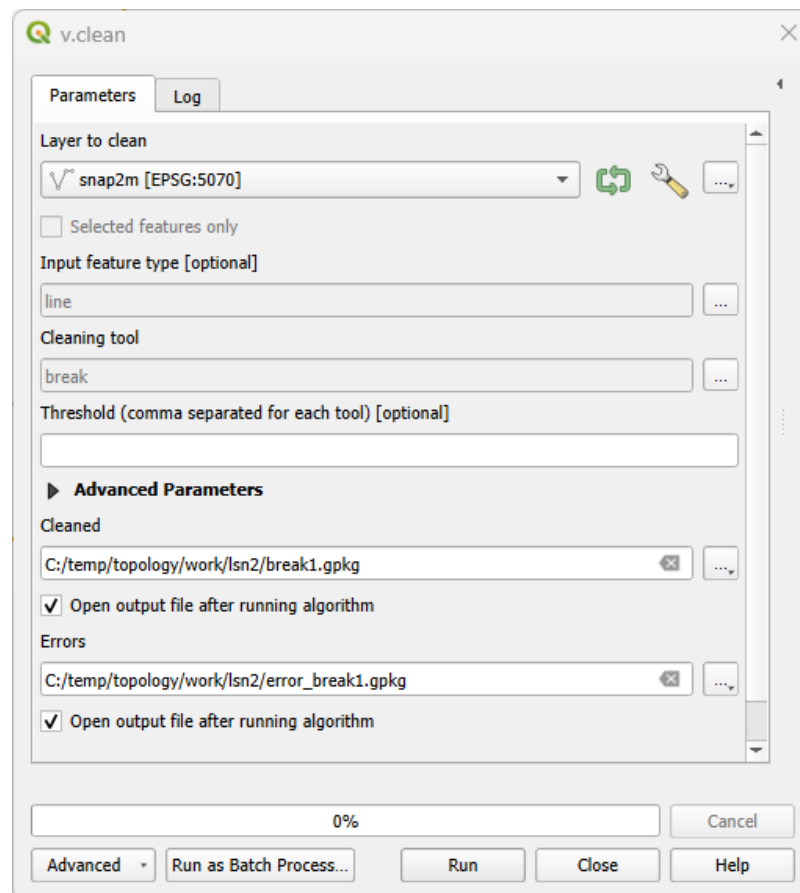


*Note that you may see warnings such as “Concurrent mapset locking is not supported on Windows” or “No attribute table found -> using only category numbers as attributes”. Ignore these messages.

The location of line segments in the output snap2m are altered slightly when v.clean is used with the Cleaning tool snap (see below). In this example, edge 146 was moved down so that the endpoint of edge 835 just touches its midpoint; thus, removing the Dangling Node error. However, a new Intersection Without Node Error (C) is introduced where the endpoint of 835 touches the midpoint of edge rid = 146.

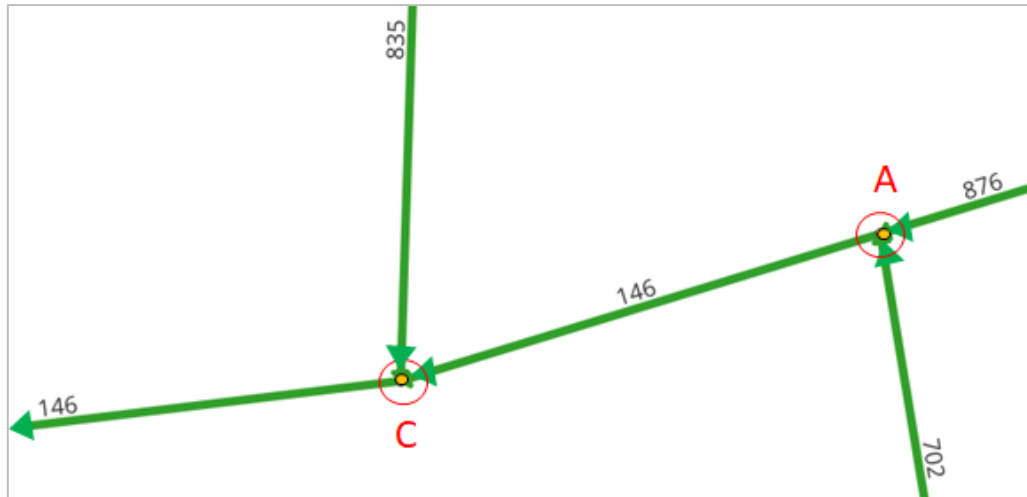


- c. **Run v.clean 2:** v.clean is used to split line features that intersect one another without a node present (e.g. rid = 146 above) using the Cleaning tool 'break'. Supply arguments to v.clean as shown below and click Run. Close the tool when it has finished.



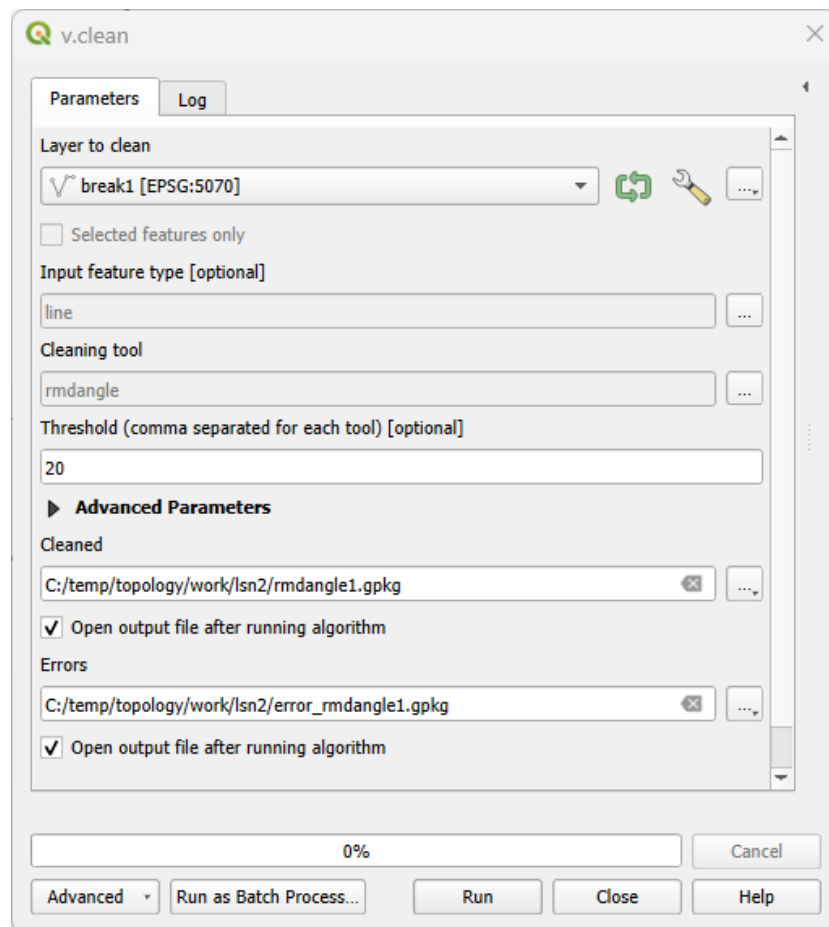
Ignore warnings mentioned in the previous step, as well as warnings that state “14 features without category were skipped...” and Output layer is empty, no features written”.

Splitting line features removes the Intersection Without Node errors in the Cleaned dataset, break1. In the example below, edge 146 was split in two, which will allow a Confluence node to be created between edges 835 and 146. Notice that the two new line features contain the same rid value as the original (rid = 146). In fact, they both contain the same attributes as the original edge. The rid values will be automatically updated the next time the LSN is generated using lines_to_lsn, but the Length field will need to be updated later in the processing workflow. The SSNbler function updist_edges can be used to do this when the argument calc_length = TRUE.

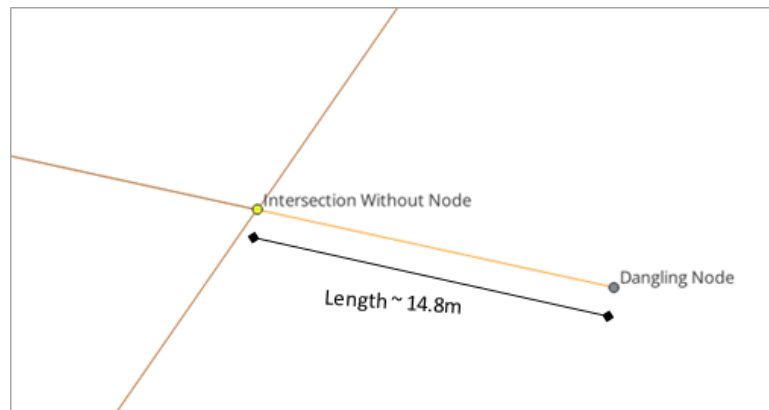


Before moving to the next step, explore the other changes that were made to edges when snap2m and break1 were created.

- d. **Run v.clean 3:** v.clean is used to remove and remaining Dangling Nodes and edges using the Cleaning tool 'rmdangle' (not 'rmbridge'). This cleaning step will also fix errors that may have been introduced in the previous two steps. Supply arguments to v.clean as shown below.

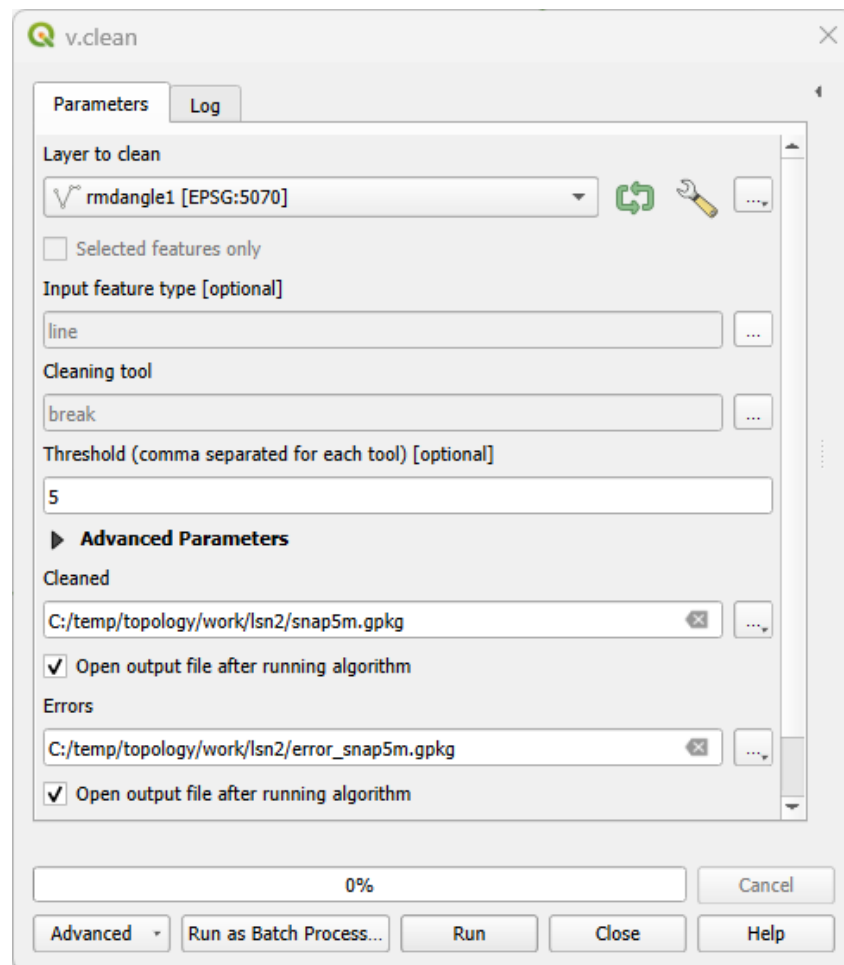


Notice the threshold value is set to 20m in this example. This value should be selected based on the length of the dangling edges in the data. It must be large enough to remove the dangling portion of the edge, but not so large that it removes headwater edge features that are shorter than the threshold value. If in doubt, be conservative. Dangling edges can be manually removed if need be.



Click Run. Once v.clean is finished, click the close button.

- e. **Run v.clean 4:** v.clean is used to snap Dangling Node errors more aggressively using the Cleaning tool 'snap'. This time, the snapping distance is set to 5m. This will remove most of the remaining Dangling Node errors in the LSN. Some will remain, but these cannot be fixed with using v.clean. Be aware that snapping at larger distances using v.clean significantly increases the risk of unintentionally deforming the edge features in the LSN. Supply arguments to v.clean as shown below and click Run. Once v.clean is finished, click the Close button.



Take a moment to investigate the differences between the Cleaned dataset, snap5m, and the input, rmdangle1, to ensure you are satisfied with the topological correction. Remember, the Errors output, error_snap5m, can be used to quickly identify and zoom into these edits and having the node_errors visible makes it obvious what the original error was. Pay particular attention to the Complex Confluence and Diverging Node that were corrected in Section 5.3 to ensure that edges were not snapped to their previous location, reintroducing those topological restrictions.

The snap5m dataset contains all the edits implemented by iteratively calling v.clean and represents the most up-to-date version of edges.

6. Remove all layers from the Layers menu before returning to R to regenerate the LSN based on snap5m.
7. Open R and run lines_to_lsn on snap5m.shp and check for topological errors, as shown below. Notice that the updated LSN is saved to a new directory called lsn3. Also notice that there has been a significant reduction in the number and type of topological errors identified in node_errors. Interestingly, there is still a Downstream Divergence. Move back to QGIS to fix the remaining errors.


```
## Import edited edges
edges2 <- st_read(paste0(lsn_path2, "/snap5m.shp"))

## Build the initial LSN and check the topology
lsn_path3<- "c:/temp/topology/work/lsn3"
edges<- lines_to_lsn(
  streams = edges2,
  lsn_path = lsn_path3,
  snap_tolerance = 1,
  check_topology = TRUE,
  topo_tolerance = 20,
  overwrite = TRUE,
  verbose = TRUE,
  removeZM = TRUE)

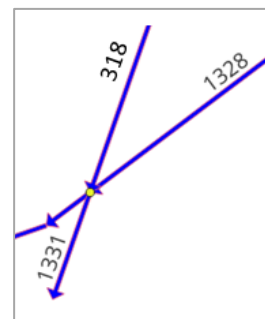
## Import node errors and format columns
node_errors <- st_read(paste0(lsn_path3, "/node_errors.gpkg"),
  quiet = TRUE) %>%
  modify_if(is.character, as.factor)

## Summarise
summary(node_errors)
```

pointid	nodecat	error	geom
Min. : 473	Confluence:3	Converging Node :3	POINT :9
1st Qu.:1315	Outlet :6	Dangling Node :5	epsg:5070 :0
Median :1330		Downstream Divergence:1	+proj=aea ...:0
Mean :1207			
3rd Qu.:1336			
Max. :1347			
NA's :2			

5.5 Correct the remaining topological errors

1. Load edges and node_errors from C:/temp/topology/work/lsn3 into QGIS. Open the node_error attribute table and inspect the error type. You should see a downstream divergence, converging nodes, and some remaining dangling nodes. We will start with the downstream divergence.
2. Open the node_errors attribute table, select "error" = 'Downstream Divergence', and zoom into the selected node_error point.
3. Change the symbology of the edges from Simple Line to Arrow, as described in previous steps. In the image to the right, the blue arrows are edges pointing in the direction of flow and the numbers are edge rid values. Notice rid = 1331, which is flowing out of the diverging node. This feature was created when the edges were split to remove illegal intersections. It was not removed with other dangling edges because its length exceeded the threshold (20m) set when rmdangle1 was created using v.clean.
4. Manually remove the dangling edge.
 - a. Select edges in the Layers Panel and Toggle Editing on.

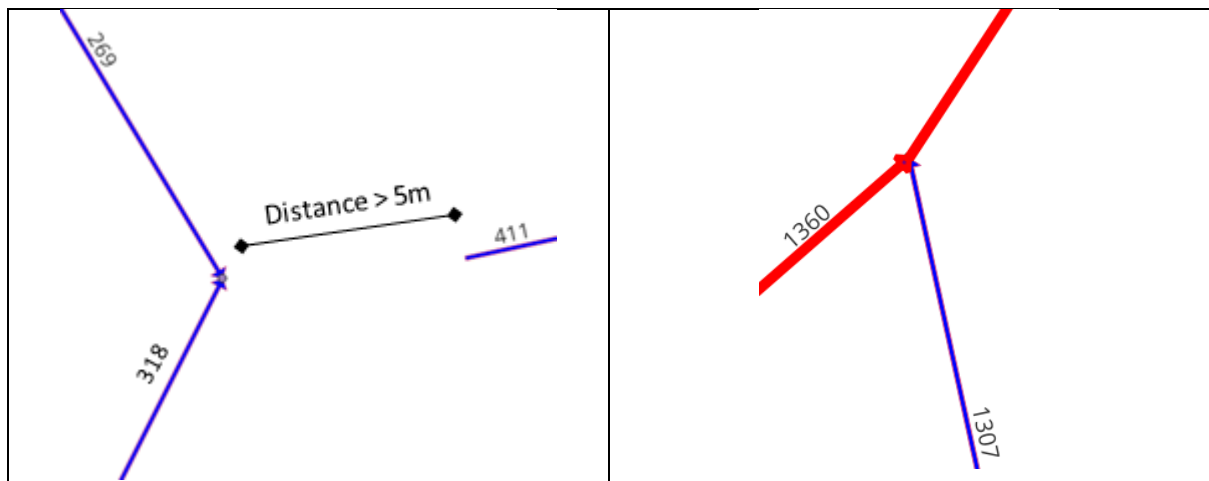


- b. Select the edge with rid = 1331 and delete it. Save Layer edits.
- 5. Examine the remaining errors in node_errors, which consist of Converging Nodes and Dangling Nodes. These remaining errors must be manually removed.

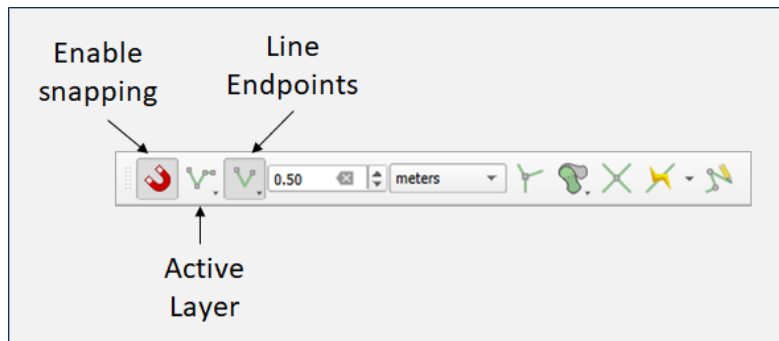
Converging nodes may appear at confluences when one of the three up or downstream line segments does not connect properly (left panel in the image below). These errors were not corrected during the previous snapping step because the distance between the stream segments was greater than 5m. As a result, they must be manually adjusted.

Most of the remaining Dangling Node errors are cases where one line feature has been snapped to the midpoint, rather than end node, of another line feature (Panel B in the image below). The solution for these errors is to break the line feature into two features (rid = 1360 below). Unfortunately, these errors must also be manually corrected.

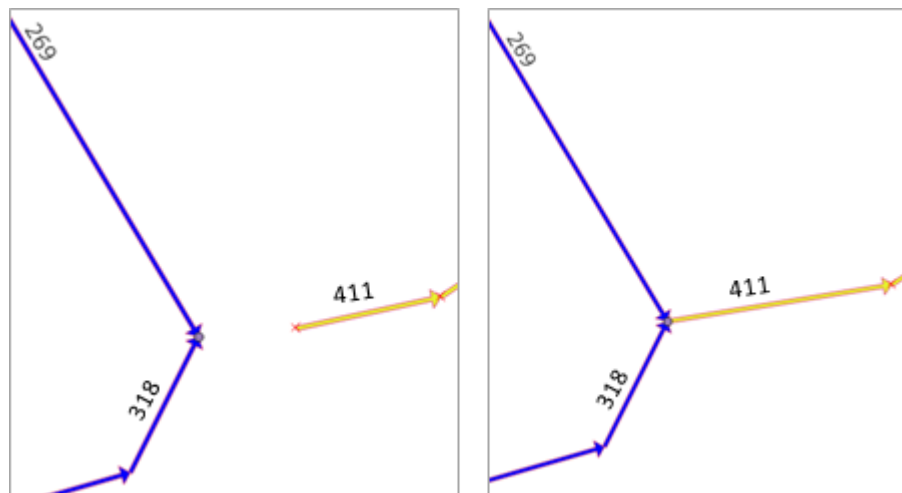
Finally, one Dangling Node was not corrected because the distance between the edge feature and the nearest line feature was greater than 5m.



- 6. Correct Converging Nodes, which are essentially unsnapped nodes in this example.
 - a. Set snapping options for editing line features.
 - i. Enable the Snapping Toolbar. Right click in the Toolbar area, scroll down, and select the box next to Snapping Toolbar.
 - ii. Enable Snapping, select Active Layer, and specify Line Endpoints. Set the snapping tolerance to 0.50 meters.



- b. Open the node_errors attribute table, select a Converging Node, and zoom into the feature.
- c. Ensure that the edges are selected in the Layers Panel and that editing is turned on.
- d. Select the edge that will be edited and click on the Vertex Tool.
- e. Hover the cursor over the end node of the edge feature you want to move. The vertices will be shown as red circles (by default). Left click once on the vertex at the end of the line to select it, move the cursor to the line feature endpoint you would like to join it to, and left click on the mouse one more time. Notice that a triangle symbol appears once you are within snapping tolerance (0.50m) of the endpoint and the cursor is automatically snapped to that location.



- f. Click on the Save Layer Edits button and repeat this process for all the Converging Nodes.
7. Correct (most of) the Dangling Nodes
- a. Open the node_errors attribute table, select a Dangling Node, and zoom in to the feature.
 - b. Identify which edge needs to be split at the intersection and select it (left panel below).


- c. Click on the Split Features tool and split the edge where the Dangling edge should join with it (right panel below).

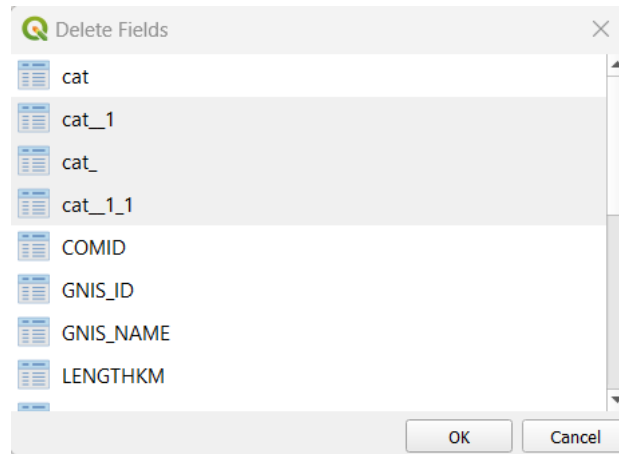



- d. Click on the Save Layer Edits button and repeat these steps for all but one Dangling Node (pointid = 1316), which is a special case described next. Note that some node_errors may relate to the same Dangling Node. Skip these errors if they have already been corrected.
8. Correct the final Dangling Node, which is a special case that requires both an edit to connect it to another edge and a split to create a location on the other edge where it can connect it to.
 - a. Zoom into the remaining node_error, which is pointid = 1316.
 - b. Select the edge you want to edit (rid = 1308; left panel below).
 - c. Use the Vertex tool to extend the line segment so that it is close to a midpoint on the nearest edge (rid = 1108; middle panel below).
 - d. Select line feature with rid = 1108. Use the Split Features tool to split it at the intersection (right panel below).
 - e. Click on Save Layer Edits and clear the selection (if one exists).

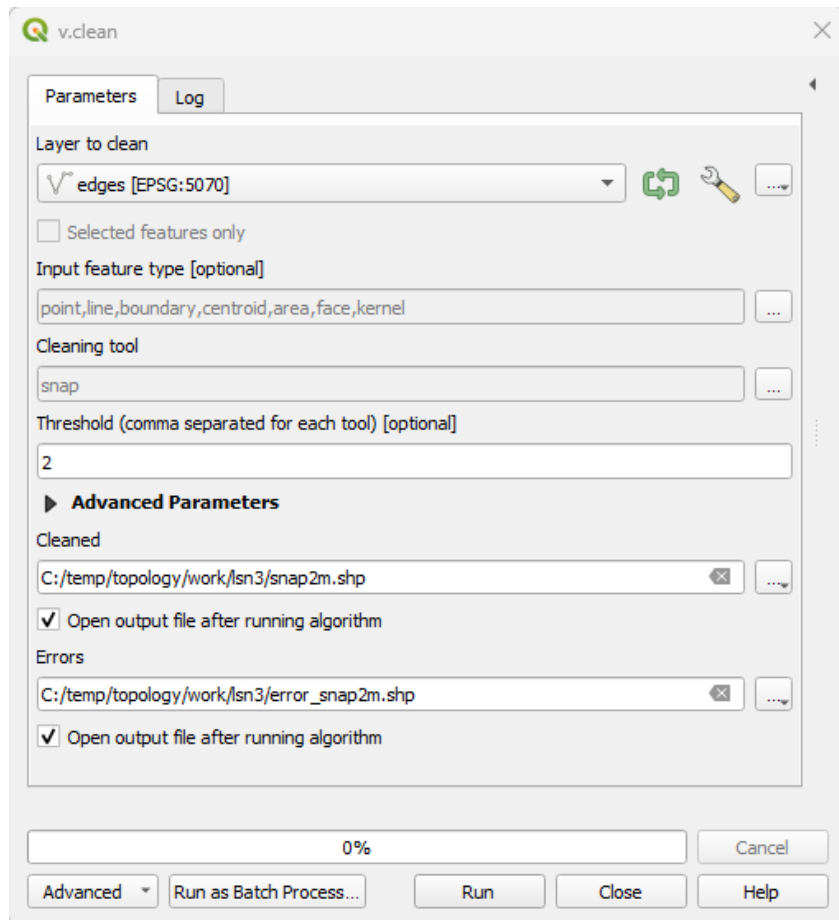


9. Remove the identifier columns added when v.clean was iteratively run (e.g. "cat_", "cat__1", and "cat__1_1"). If they are not deleted at this stage, they will cause problems in the following editing steps.

- a. Open the attribute table for edges.
- b. Click on the “Delete field” button, , in the menu ribbon to open the Delete Fields tool.
- c. Select the three extraneous fields as shown below and click OK.



- d. Click the “Save edits” button  in the menu ribbon and the Toggle editing mode button to stop editing. Close the attribute table.
10. The v.clean tool is now run a fifth time to snap the nodes we edited in the last steps.
- a. First, open and run the v.clean tool with a threshold of 2m to snap the unsnapped nodes. Pass in arguments as shown below.



11. Remove all the layers from QGIS and return to R.
12. Import the edges that were edited in the last step (snap2m) and rerun the lines_to_lsn function to build a new LSN and check the network topology, as shown below.

```
## Import edited edges
edges3 <- st_read(paste0(lsn_path3, "/snap2m.gpkg"))

## Build the initial LSN and check the topology
lsn_path4<- "c:/temp/topology/work/lsn4"
edges<- lines_to_lsn(
  streams = edges3,
  lsn_path = lsn_path4,
  snap_tolerance = 1,
  check_topology = TRUE,
  topo_tolerance = 20,
  overwrite = TRUE,
  verbose = TRUE,
  remove_ZM = TRUE)
```

The output messages in the console returns “No obvious topological errors detected and node_errors.gpkg was NOT created.” In this case, we do not need to manually check for or correct errors before continuing because we’ve done that throughout this tutorial.

Congratulations! The LSN is error free and ready for subsequent processing steps using SSNbler.

6. References

QGIS Development Team (2024) QGIS Geographic Information System. Open Source Geospatial Foundation Project. Version 3.28.7. Available at: <https://qgis.org>

R Core Team (2023) R: A Language and Environment for Statistical Computing. R Foundation for Statistical Computing, Vienna, Austria. <https://www.R-project.org>