

# Topology extension for OpenJUMP

v 0.5 (2012-06-26) by Michaël Michaud

---

## 1.1 General concepts

Topology Extension is a set of plugins bundled as a jar file to be placed in OpenJUMP plugin directory (default plugin directory is lib/ext).

All plugins from Topology extension are related to topology error detection and correction.

Important : this extension works on geometries and heavily uses JTS capabilities. On the other hand, graph extension uses JGraphT capabilities and works on graphs built from feature relations.

This extension has no external dependency, but JTS and OpenJUMP.

### 1.1.1 Topology extension plugins and common options

Every plugin from topology extension is installed in the Plugins menu of OpenJUMP, in a submenu called... Topology.

Most options have a tooltip giving more details about option signification.

## 2 Remove micro-segments Plugin

This plugin removes micro-segments from LineStrings or Polygons.

This process is often needed before a cleaning process involving topology.

Indeed, most cleaning process use a distance threshold to decide if two features (resp. nodes, segments) should be snapped together, and presence of micro-segments less than this threshold inside single geometries often disrupt the process.

What the plugin do and what it doesn't do :

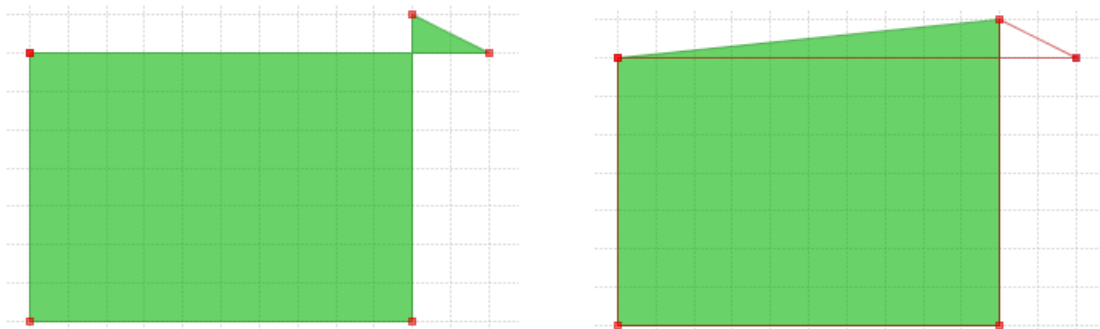
- "Remove micro-segment" carefully chooses the point to remove from the geometry so that the deformation is minimal : micro-segments have two vertices. Vertices located on the geometry boundary are never removed, and if both vertices are strictly inside the linestring, the plugin removes the vertex where interior angle is the closest to flat angle ( $180^\circ$ );



- "Remove micro-segment" processes only LineStrings and Polygons yet. GeometryCollections are just ignored.
- "Remove micro-segment" does not check if the vertex it will remove is used by another geometry. The process, which is purely geometric, may break topology consistency. Hopefully, other topologic correction tools should be able to repair what has been broken.

Special use-cases :

"Remove micro-segments" is able to fix invalid polygons with a very small self-intersection :



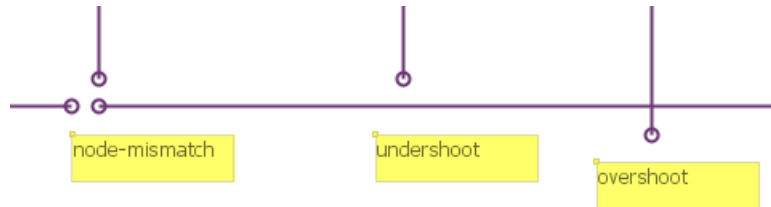
Note that the problem is fixed by the removal of a point, which only works if self-intersection is made of a single micro-segment.

Currently, OpenJUMP lacks a user-friendly-tool to fix larger self-intersection problems. A quite complex process could be to create linestrings from polygon boundaries, node those linestrings, polygonize the result, remove polygons representing holes, get attributes back using matching tools...

### 3 Network topology cleaning Plugin

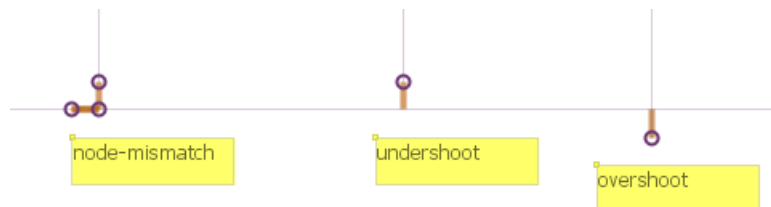
This plugin can process a dataset representing a network, and detect and/or fix some common topologic errors like

- node mismatches : nodes closed to each other but not exactly the same
- undershoot : a linestring end is close to another linestring but is not snapped on it



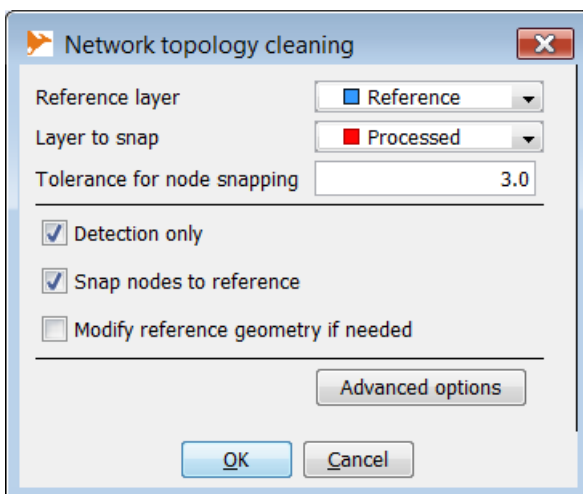
- overshoot : a linestring crosses another one and end just after the intersection

Formal definition : network topology cleaning plugin finds every node of a network (end-points of the network linestrings) which are closed enough to another LineString (closed-enough being user-defined), but not snapped on this LineString (no common vertex).

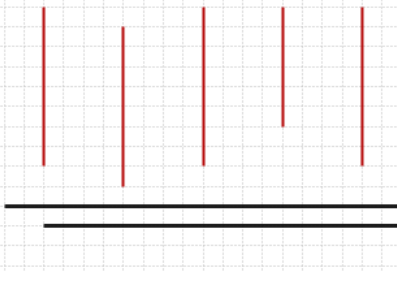
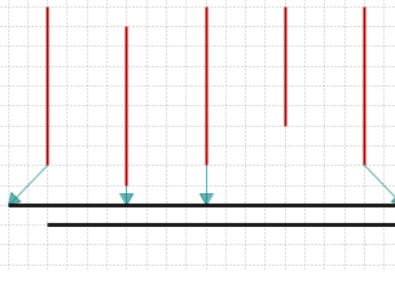
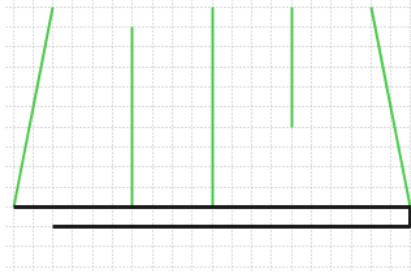


Anomalies can be either detected or corrected.

#### 3.1 The network topology cleaning plugin main parameters



The main parameter is the distance threshold between a node of the layer to process and a feature of the reference layer :

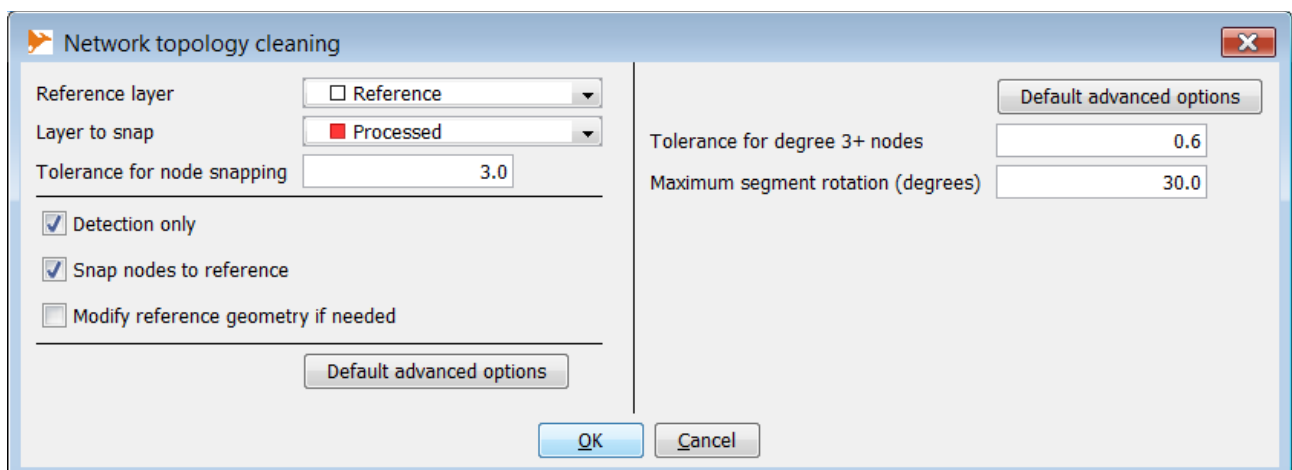
		
<p>Tolerance is 3 meters. The 1st, 2nd, 3rd and 5th feature are under the tolerancere.</p>	<p>Detection :</p> <p>1st feature is at less than 3 m from a reference node</p> <p>2nd and 3rd features are at 1m from a reference segment, at 2 meters from another one and at more than 3m from any vertex</p> <p>4th feature is at more than 3 meter from the reference feature</p> <p>5th feature is at less than 3 m from a reference vertex</p>	<p>Correction :</p> <p>1st and last features are properly snapped to reference</p> <p>2nd and 3rd features are snapped, but a vertex needed to be inserted into the reference layer.</p>

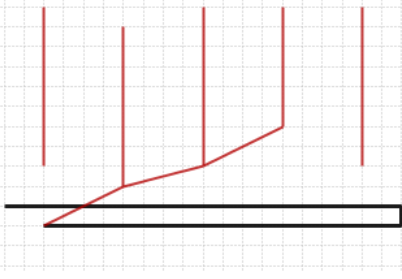
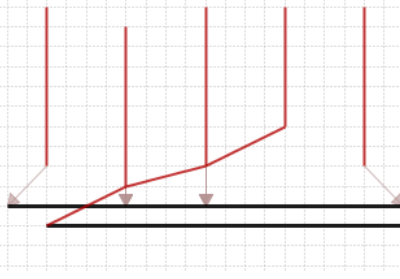
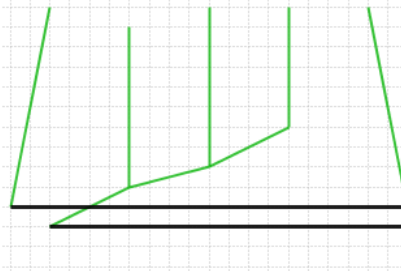
## 3.2 Advanced options : degree 3 nodes

An option is available to deal with degree 3+ nodes of the processed layer. It maybe useful to decrease the tolerance parameter for these nodes, because if they are not snapped on the reference layer, there is a higher probability that it should not be snapped.

Default value of degree 3 nodes snap tolerance is 1/5 base tolerance.

Changing the tolerance for degree 3+ nodes or setting it back to default is quite easy :


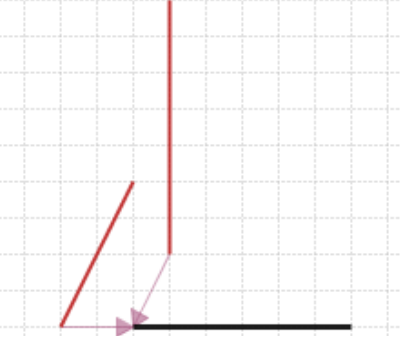



		
Degree of the lower node of vertical features 2, 3 and 4 = 3 (this means that feature extremities not only snap another feature, they share their node with another feature node)	Mismatch indicators are created for every node at less than 3 m from the reference feature but...	only the degree 1 nodes are snapped. Degree 3 nodes would have been snapped if they were under the default tolerance (here, $3/5 = 0,6$ m)

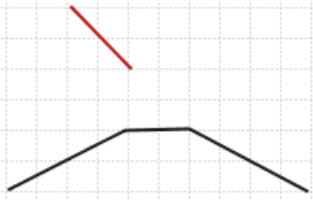

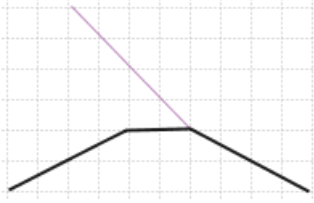
### 3.3 Advanced options : rotation parameter

There is a second editable parameter which is a rotation tolerance.

Rotation tolerance is available for pendant nodes only (degree 1). Indeed, using a rotation tolerance on degree 2+ nodes would have bad side effects as it could break many connected features just because of their segment orientation.

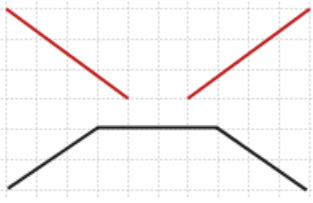
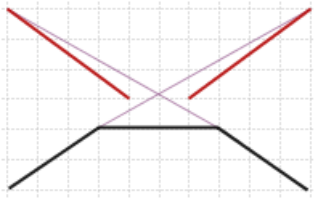
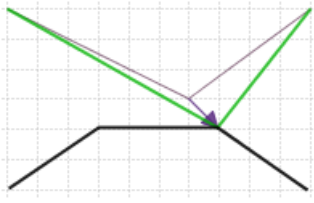
		
Here, we have two candidate features at the same distance from the reference layer : 2 m	Both could be snapped to the reference, but for the first one, the snap will rotate the segment much more than for the second one	Finally, with the rotation tolerance set to $15^\circ$ , the first feature is not snapped while the second one is.

When several vertices are found within the distance tolerance rotation criteria is used to determine which vertex to snap to

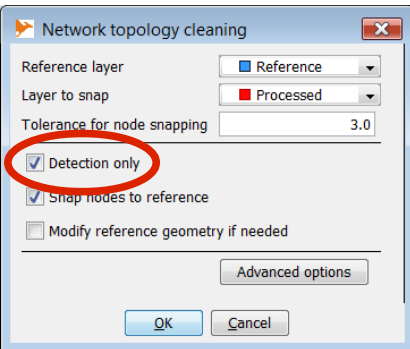
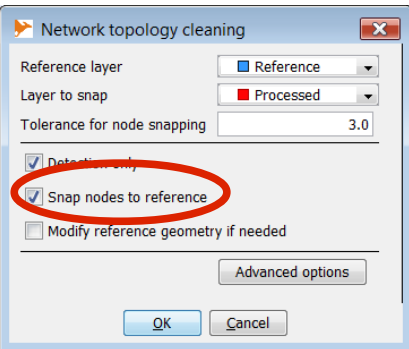
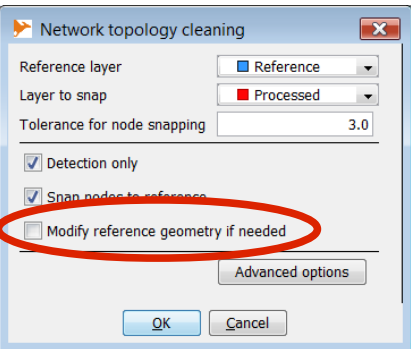
		
The red LineString is within the 3 m tolerance and both vertices of the nearest reference segment are in the 3 m tolerance.	The choosen snap vertex is not the nearest one...	but the one introducing the smallest segment rotation.

### 3.4 Corner cases

Corner cases and situations where the plugin may give weird results

		
Because of the angle criteria preference, in this case where both vertices are within the distance tolerance (4 m) and where segment rotations are minimized...	the process results in two crossing lines.	A work-around is to pre-process the input layer to eliminate the node-mismatch, then to process the result.

### 3.5 Output options

		
Detection : a layer will be created with translation vectors needed to snap nodes to reference features. See below this layer attributes	Correction : a copy of the processed layer is done and nodes of this copy located near but not snapped to the reference layer will be	Reference correction : this option is only available if the reference layer is editable. If checked, missing vertices will be inserted in the reference

	snapped.	layer just where the processed feature has been snapped to.
--	----------	---

---

### 3.6 Mismatch vector layer

---

**SNAP attribute** shows if the node has finally is been snapped on a reference node, on a reference vertex, on a reference segment, or not snapped at all :

**Snap to node** : the node has been snapped on a reference node

**Snap to vertex** : the node has been snapped to a reference vertex

**Snap to segment** : node has been snapped in the middle of a reference segment

**Not snapped** : the node has not been snapped

**Not snapped (D > xx)** : the node has not been snapped because it is over the distance tolerance (the degree 3+ one, otherwise, there is no reason to have a mismatch vector)

**Not snapped (A > xx)** : the node has not been snapped because the result would have been over the rotation angle tolerance

For snapped nodes, **attribute "Rotation"** gives the angle the node segment has been rotated to snap the reference feature (in degrees).

## 4 Project points on lines PlugIn

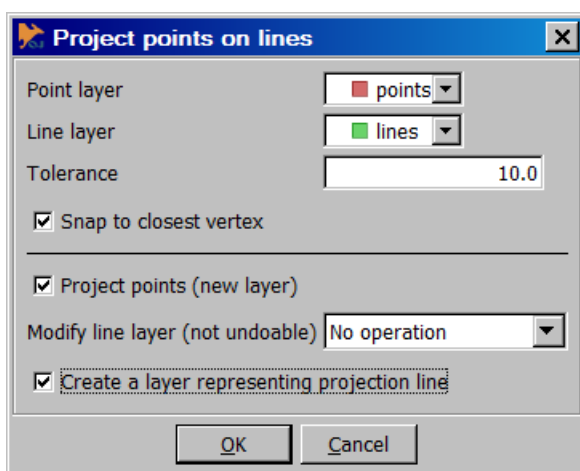
This plugin project points of a point layer onto lines of a line layer. Optionnaly, projection points can be inserted into lines or used to split the lines.

### 4.1 Inputs

**Point layer** must contain at least one point.

**Line layer** must contain at least one LineString. MultiLineStrings will not be used. If you have MultiLineStrings in your linear network, decompose them into simple LineStrings before starting the process.

### 4.2 The PlugIn parameters



**Tolerance** : This is the main parameter. If the distance to the nearest line is greater than tolerance, the point will not be projected. By default, the point is projected to the closest location (which generally means at right angles to the closest segment). See above for snap option.

**Snap** : if checked, the algorithm will measure the distance between the point and the vertices located immediately before and after the orthogonally projected point. If the closest of these points is under the tolerance, the point will be projected on this vertex. Note that if the point is already on the line, but near a vertex point, the snap option will move it along the segment to the closest vertex.

	Without snap option, points are projected at right angle to the closest line, except if their distance to the line is greater than the <b>tolerance</b> (here, tolerance = 3 units).
	Now let see if <b>snap</b> is activated and the reference line has vertices near the projection points :
	<ul style="list-style-type: none"><li>1 : project to the closest vertex (which distance is less than the tolerance)</li><li>2 : project to the closest vertex which happens to be also the closest location on the line.</li><li>3 : project to the closest vertex (&lt; tolerance)</li><li>4 : project orthogonally (closest vertex is &gt; tolerance)</li><li>5 : not projected (&gt; tolerance)</li></ul>



**Project points (new layer)** : create a new layer containing the projection of points on the nearest line. The layer is called "sourceLayerName - projected". It has the same schema as the source point layer plus one attribute containing distances between source point and projected point.

**Modify line layer (not undoable)** : to be more safe, this component is only available if line layer is editable. It has three options :

- no operation : do not modify line layer
- insert vertices in lines : add the projected points to the line geometry
- split lines : split line where points have been projected

Inserting points and splitting lines is performed in the source layer itself and is undoable. This choice has been guided by performance and memory considerations. So, make a copy of your layer before processing it if you're not sure.

**Create a layer representing projection line** : create small line segments between source point and projected points. Moreover, for further analysis, these lines have the same attributes as the points, plus the distance between the point and its projection (i.e. the segment length).

## 5 Coverage cleaning Plugin (to be done)

This plugin cleans a coverage layer with small defects like overlaps and/or gaps.

It is based on Java Conflation Suite (mainly coded by Martin Davis, JTS architect), released by Vividsolutions. The only improvements added from the original plugin are

small performance improvements

full support of polygons with holes

**Attention : ça ne semble pas marcher dans le cas où les contours limitrophes ne sont pas dans le même sens (ex. un contour extérieur dans le sens cw et l'autre dans le sens ccw).**

How the plugin works ? The plugin tries to match each segment of each polygon with segments of neighbour polygons. Segments match when :

- they are not topologically equals (not equals and not equals with opposite segment)
- the minimum distance between segments is less than the user-defined tolerance,
- the angle between both segment is less than the user-defined tolerance
- the orthogonal projection of each segment on the matched one is non null

If matched segments are found on a contour, matching contours are merged by :

- snapping vertices of matched segments when they are close enough
- inserting missing vertices in each ring (a vertex is missing when a point of a matched segment has no matched vertex)

