An alternative method for automatic coding of stream order on digital stream network data

Gido Langen
Surveyor General Branch
Natural Resources Canada
Gido.Langen@NRcan.gc.ca

Knowledge of stream order is necessary for appropriate watershed management techniques, and a new method for automating stream order in digital hydrographic data sets is presented here. This approach uses a relational database and geographic information system (GIS) along with structured query language (SQL) to automatically code a digital hydrographic database to Strahler stream order. The procedure was successfully tested on a hydrographic database from the Province of Alberta, Canada containing nearly one million stream segments. The method ran error-free using significantly less processing time than other alternative approaches, and can be applied to other stream data sets.

Keywords: stream order; automated stream order coding; watershed management; hydrographic data

1. Introduction

Resource management and, in particular, water resource management requires a good understanding of watershed issues. Addressing these issues usually involves use of hydrographic data such as stream and lake features, which have been captured in spatial databases. From these hydrographic data, we can derive stream network data, composed of stream centerlines and arbitrarily placed flow lines through other features such as lakes. While their usefulness is somewhat limited, it is the information that is stored with these data sets such as stream flow or particulate matter that makes them useful when investigating watershed issues. Collecting that information, however, is expensive. Instead, one can group stream network data on common characteristics so that correlation with land use concerns, such as the impact of land use on fish habitat and changes in erosion potential, can be estimated. A widely adopted method of stream network classification is Strahler's stream ordering (Strahler 1952, 1957), which has been applied to stream network data throughout the world.

Many organizations apply stream ordering to manage and assess natural resources. This can reflect the different landscape and resource characteristics along waterways as illustrated by correlating water chemistry parameters with Strahler-Order coded streams (Dodds and Oakes 2008), and by using Strahler-Order coded streams for better forest management (Prepas et al. 2008). As physical and biological changes occur in streams from the headwaters to downstream sections as predicted from the river continuum concept (Vannote et al 1980; Johnson et al. 1995; Mourier et al. 2008), knowledge of stream order is

important to aquatic ecology and watershed management. Furthermore, stream ordering is a more cost-effective method than classifying streams on the basis of channel morphology (Rosgen and Silvey 1996).

Coding stream network data according to Strahler's methods can be an extensive effort in itself when coding the data interactively. Only a few companies provide automated Strahler-Order coding services such as RivEX (Hornby 2010), or software tools (ESRI 2010, Safe Software 2010) which use an algorithm developed by Gleyzer et al. (2005). However, this method may not be easily accessible to most readers who lack a good understanding of set theory, and it may be rather difficult to implement without strong programming skills.

Miller et al. (1996) used the topological relationship of upstream and downstream line segments, but their method is rooted in the procedural programming paradigm (i.e. specifying the steps the program must take to reach the desired state) as opposed to using only relational database techniques. Finally, Whitaker et al. (2002) used the topological relationships of upstream and downstream line segments for automated stream leveling but did not extend the approach to stream ordering. Neither approach used a purely relational database driven technique which we do in our approach. This article describes a simple but effective method to automatically apply Strahler-Order codes to stream network data. It only requires updating of attribute values and, moreover, no spatial data editing is needed.

2. Methodology

2.0 Prerequisites

Any stream network data used with the described method of Strahler-Order coding must be fully topologically structured (see Appendix B). It must be one-directional, flowing from source to destination with each line segment having a unique identifier and start and end nodes. The stream network may be braided having divergent-convergent stream segments; circular flow, however, is not permitted.

2.1 Sample Data Set

Figure 1 illustrates a small abstracted sample of a stream network to demonstrate automated Strahler-Order coding. Streams in Figure 1 flow generally from the southwest to the northeast, and belong to three levels of Strahler-Order: 1, 2 or 3. Strahler Order is a form of stream ordering based on accumulation. Classification begins with First Order at the headwaters of the stream network and increases in Strahler Order class towards the mouth. Every confluence of streams of the same order will increment the Strahler Order downstream except where they arise from the confluence of braided streams. Confluences between streams of higher and lower order result in the higher stream order continuing downstream without any increase in the Strahler-Order value.

2.2 Methods

We use an iterative method of elimination as illustrated in Figure 2 to assign Strahler-Order values. In the first iteration, we locate the outermost branches of a stream network applying rules that we describe later in the article. This

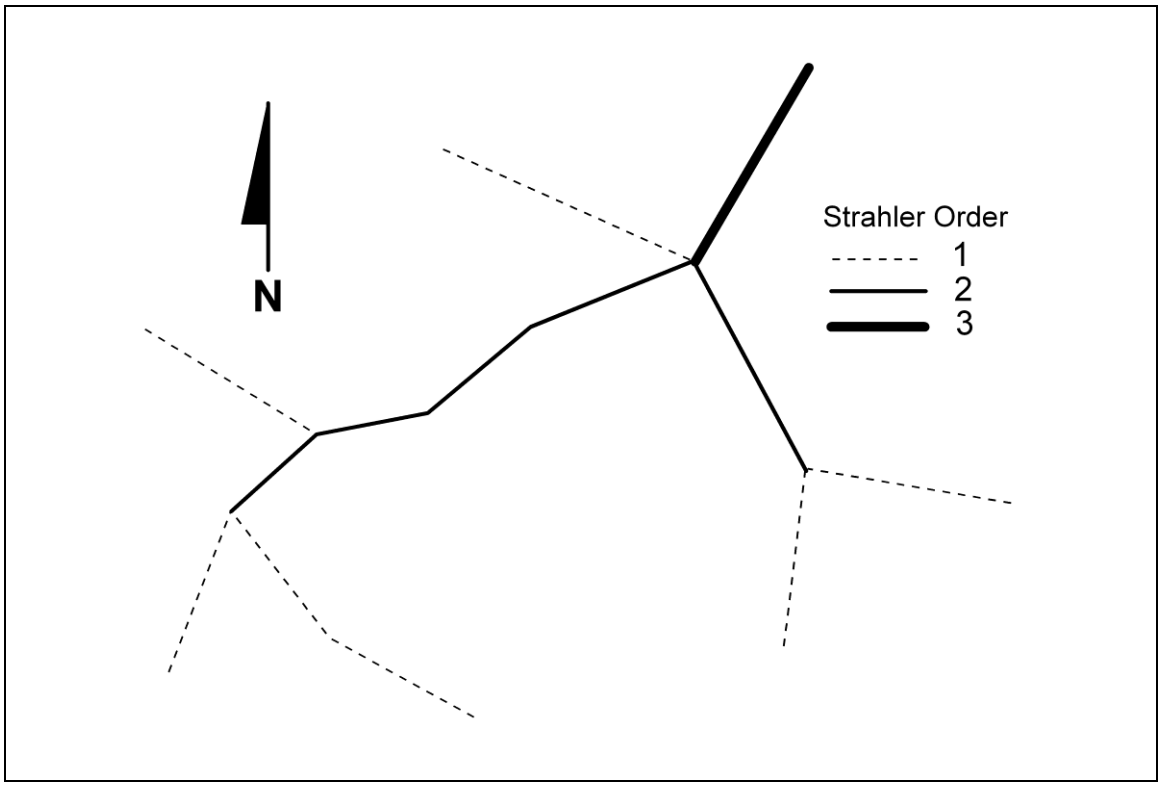

Figure 1. Sample stream network with 3 levels of Strahler-Order coding

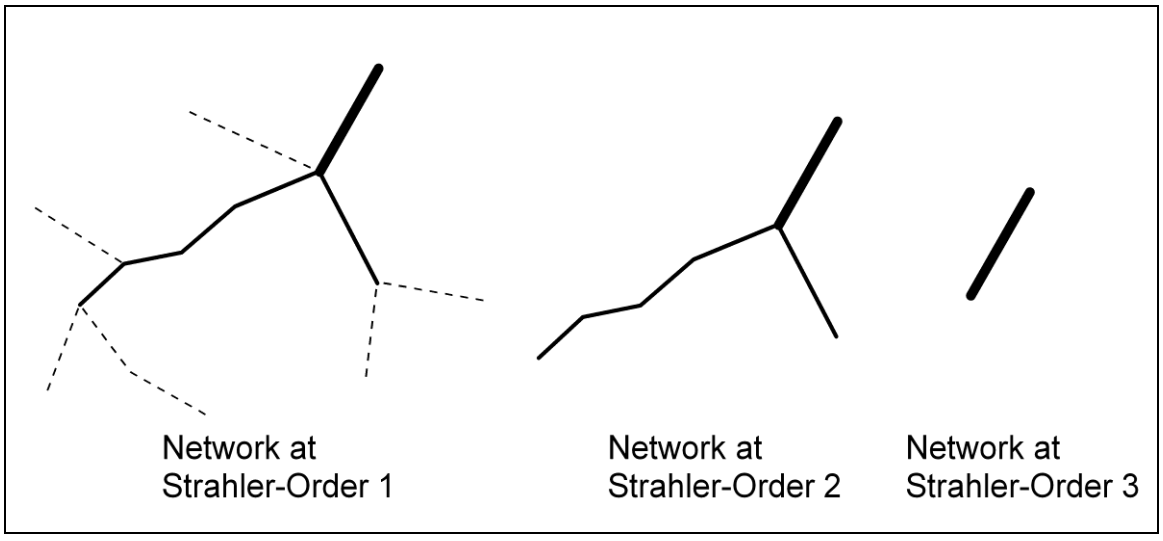

Figure 2. Iterative elimination of branches

approach is opposite to that taken by Whitaker et al. (2002) who automated stream leveling starting at the terminus rather than the headwaters. Once the outermost branches have been identified, we set their Strahler-Order values to 1, and then eliminate them from the network, with the remaining uncoded branches unchanged. We mimic elimination of branches using the Strahler-Order values where a Strahler-Order value of 0 signifies presence of uncoded stream segments, while a value larger than 0 conveys absence of uncoded stream segments (that is the already coded segments). In the second iteration, "new" outermost branches remain after "eliminating" the already coded first-Order branches. We locate them applying the same rules as in the first iteration except that their Strahler-Order value is set to 2 after they have been identified. Similarly, the Strahler-Order value of the last remaining branch is set to 3 after the third iteration.

Personal anecdotal evidence suggests that most organizations that manage natural resources usually lack the programming experience to develop automated Strahler-Order coding programs, or the funding to acquire software (Safe Software 2010) for that purpose. GIS software, on the other hand, is usually at hand including Relational Database Management System (RDBMS) technology. Most GIS users are familiar with Structured Query Language (SQL) commands (Codd 1970) required to operate RDBMS software. In this article, we provide an SQL solution to automate Strahler-Order coding.

2.2.1 Data structure

In order to solve Strahler-Order coding of streams automatically, we first represent the sample network by segments and nodes (Figure 3), and store this representation in two tables: ***Segment*** and ***Node*** (Figure 4).

We name the segments A through M and number the nodes 1 through 14. Nodes play an important role in the ***Segment*** table: They define a segment's direction by serving as either *Start-node* or *End-node*, and frequently function simultaneously as both *Start-node* and *End-node*. For example, *Node* 2 serves as *Start-node* of segment C and *End-node* of segment D.

2.2.2 Automation rules

The relationship between segments and nodes forms the basis for locating particular segments. Our method dictates that we find the outermost branches of the stream network in each iteration. A branch, though, may consist of more than one segment; for example, segments C and D form a branch with only D being a source segment. First, we determine all source segments which share one characteristic: Their *Start-nodes* have no corresponding *End-nodes*.

> Rule (1) specifies that we find all segments whose *Start-nodes* have no corresponding *End-nodes* and set their Strahler-Order code to the current Strahler-Order value. For example, node 13 serves as *Start-node* of source *Segment* D but never functions as *End-node*.

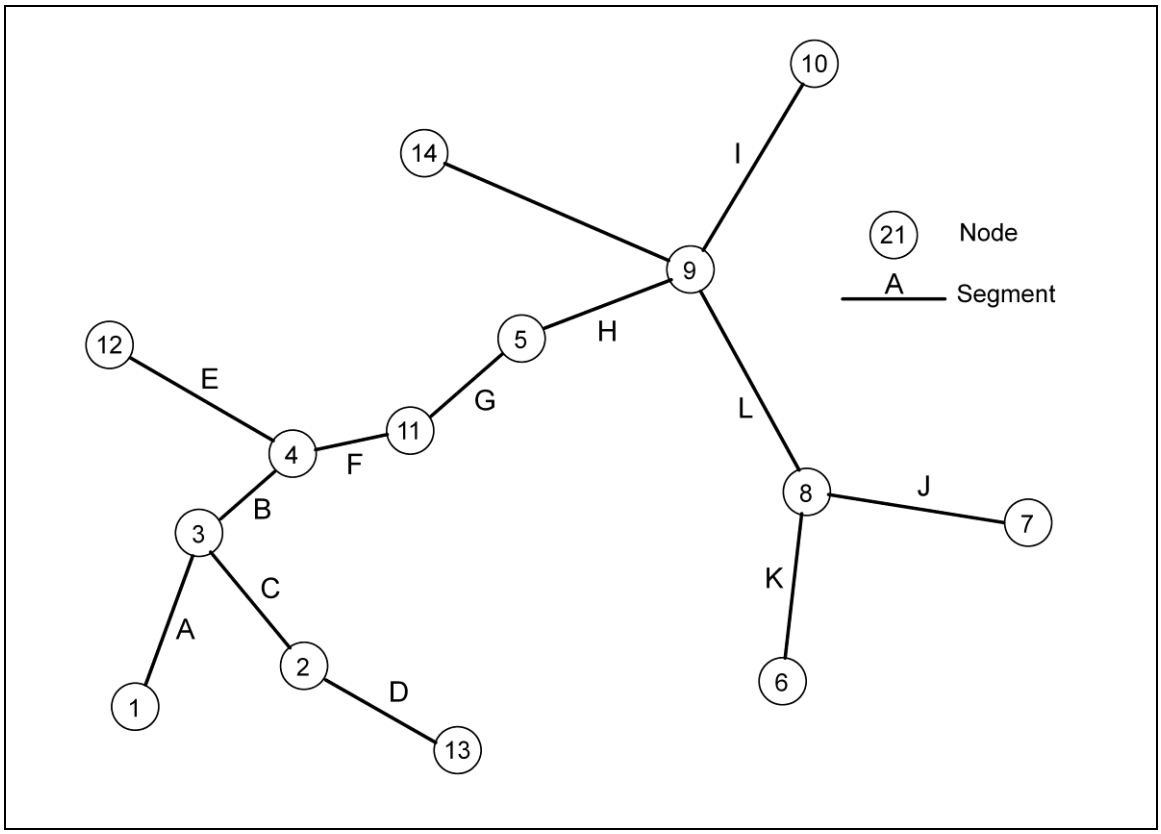

Figure 3. Graphic representation of sample stream network

Segment	*Start-node*	*End-node*	*Code*
A	1	3	0
B	3	4	0
C	2	3	0
D	13	2	0
E	12	4	0
F	4	11	0
G	11	5	0
H	5	9	0
I	9	10	0
J	7	8	0
K	6	8	0
L	8	9	0
M	14	9	0

Node
1
2
3
4
5
6
7
8
9
10
11
12
13
13

Figure 4. Database tables Segment and Node

The first rule suffices to find all source segments, while all other segments remain uncoded. We still need to find all segments within source branches that are downstream from source segments. For example, segments C and D form a complete source branch of which only source segment D has been coded so far. They may have been split because they represent different stream characteristics: Segment C may represent a perennial stream while segment D may represent an intermittent stream. The question arises: "How we can find the remaining segments in source branches?" We could locate all downstream segments which would locate segment C but also segment B. They differ in that segment B originates from a confluence of multiple upstream segments while segment C starts at a node that merely forms a bridge between two segments. That rationale, however, begs a second question: "How do segments C and G differ?" Both are downstream from a single upstream segment. But only segment C is downstream from an already coded upstream segment. We apply both characteristics in a second rule.

> Rule (2) specifies that we find all segments downstream from source segments so long as their connecting nodes are merely bridges between two connecting segments and the upstream segment has already been Strahler-Order coded.

The second rule in itself is applied iteratively until the source branch ends in a node that represents a confluence. A third rule comes into play to determine braided streams which start with divergent stream segments. In terms of the topological model, braided streams start at nodes that are used more than once as start nodes.

> Rule (3) specifies that we find all nodes that are the Start-node of more than one segment. We apply the second rule to find all downstream segments within braided branches and code them as braided segments.

A fourth rule is required to account for confluences arising from braided streams to prevent increasing the downstream Strahler-Order.

> Rule (4) specifies that confluences arising from braided streams keep the highest Strahler-Order code of the upstream converging stream segments.

Rules (3) and (4) are also applied to braided streams within braided streams.

2.2.3 Implementation

We employ relational database techniques to implement our method. First, we find all source segments – segments with *Start-nodes* that never occur as *End-nodes*. We start by determining all unique *End-node* values in the network as shown in the ***Unique-end-node*** table, and then find all the corresponding *Start-node* values in the ***Segment*** table that cannot be matched with *End-node* values

	Segment	Start-node	End-node	Code
●	A	1	3	1
	B	3	4	0
	C	2	3	0
●	D	13	2	1
●	E	12	4	1
	F	4	11	0
	G	11	5	0
	H	5	9	0
	I	9	10	0
●	J	7	8	1
●	K	6	8	1
	L	8	9	0
●	M	14	9	1

End-node
2
3
4
5
8
9
10
11

Figure 5. Segment selection where Start-nodes are not among nodes in Unique End-node Table

	End-node	Frequency
●	2	1
	3	2
	4	2
●	5	1
	8	2
	9	3
●	10	1
●	11	1

Figure 6. Frequency of End-node values

in the ***Unique-end-node*** table. The resulting six records are highlighted with bullets in the ***Segment*** table in Figure 5; they become the source segments. We enter the value 1 in the *Code* field.

For example, node 1 occurs once as *Start-node* of segment A and never as *End-node* of any other segment. Therefore, segment A becomes a source segment and is assigned Strahler-Order code 1.

Second, we determine segments downstream from source segments up to the first confluence. These segments are still uncoded but should receive the current Strahler-Order code (cf. segment C in Figure 2). We know from the discussion above that these segments have a *Start-node* value that occurs only one additional time as the *End-node* value of corresponding upstream segments. We can find these nodes by summarizing the *End-node* values of all segments and examining their frequency as listed in the ***Frequency*** table in Figure 6.

Only *End-node* values with a frequency of 1 bridge upstream and downstream segments, resulting in four candidates marked with bullets in Figure 6. These are candidates across the entire stream network. Only nodes that form a bridge to segments that have already been coded are desired – in our sample, we only want *End-node* 2. We can find the desired *End-node* values applying the following rationale: we require all *End-node* values that have a frequency of 1 in Figure 6 and whose corresponding segments have already been assigned the current Strahler-Order code. Only segment D has an *End-node* value that matches those marked in Figure 6 whose Strahler-Order code value is already 1 as depicted in Figure 7.

We can now search for downstream segments. We determine downstream segments when matching their *Start-node* values to *End-node* values of the currently selected segments as shown in Figure 8. In our example, only *Start-node* 2 of segment C will match *End-node* 2 of segment D. We set its Strahler-Order code value to 1.

We invoke the second rule only until a node is encountered that represents a confluence that does not originate from converging braided segments. Segment C ends with node 3 which represents a confluence, which completes coding of Strahler-Order level 1. At the second level of Strahler-Order coding, Segment B, F, G and H constitute a branch. We apply rule 1 once to find the new source segment B, but apply rule 2 repeatedly to capture segments F, G and H in cascading sequence until we encounter node 9 – the confluence of segments H, L and M.

Third, we determine braided stream segments. We summarize the *From-nodes* values of all segments, examine their frequency, and find that node 11 occurs twice as *Start-node* identifying it as the beginning of braids. The downstream segments, G and N, are coded as “braided” (in our implementation receiving a

Segment	Start-node	End-node	Code
A	1	3	1
B	3	4	0
C	2	3	0
D	13	2	1
E	12	4	1
F	4	11	0
G	11	5	0
H	5	9	0
I	9	10	0
J	7	8	1
K	6	8	1
L	8	9	0
M	14	9	1

End-node	Frequency
● 2	1
3	2
4	2
● 5	1
8	2
9	3
● 10	1
● 11	1

Figure 7. Selection of segments through bridge nodes

Segment	Start-node	End-node	Code
A	1	3	1
B	3	4	0
C	2	3	1
D	13	2	1
E	12	4	1
F	4	11	0
G	11	5	0
H	5	9	0
I	9	10	0
J	7	8	1
K	6	8	1
L	8	9	0
M	14	9	1

Figure 8. Determining downstream segment C

value of “1”). Forth, we propagate the Strahler-Order value “2” of segments G and N to segment H because it is downstream from a confluence of braided segments (and, therefore, does not increase the Strahler-Order value).

3. Application

We applied our method to the stream network data set of the Province of Alberta, Canada, which is fully topologically structured (Jaques 2007). The data reside in an ESRI Spatial Database Engine database and are provided in ESRI formats (ESRI 2010). The complete data set includes almost 1,000,000 segments covering seven major drainage basins from Strahler-Order level 1 up to 10. The Province of Alberta required that all streams be Strahler-Order coded. A manual approach was not considered feasible as it would have required months to complete. In addition, it would have required a thorough quality control process. By contrast, our database driven technique required merely a single day (24 hrs) of computer processing on a standard PC (3.0 GHz chip, Pentium 4 CPU) running a common database software. Although the algorithm is airtight, a visual, color-coded quality check of the output was done; no errors were discovered. The actual Structured Query Language (SQL) code is simple (and does not require any advanced database technology training) (see Appendix A). Furthermore, the database script runs error free. No additional quality control is required once the program completes, assuming that the input data are fully topologically structured, have no unintended gaps, and possess correct flow direction. This method can also be used for checking the integrity of the input data prior to Strahler-Order coding them in the first place.

4. Conclusions

A straightforward procedure using a relational database in a GIS environment was applied to a digital stream network data set and successfully coded all stream segments to their correct Strahler stream order. This method is easier to implement then programming, and does not require accuracy checking if the original dataset is error-free. The SQL script will be used for watershed management, river ecology and hydrologic applications in Alberta, Canada, and can be used on any topologically structured digital stream data set.

Acknowledgements

This article is based on work that was completed for a project funded by the Government of Alberta, Department for Sustainable Resource Development, Resource Information Management Branch.

References

Codd, E.F. 1970. A Relational Model of Data for Large Shared Data Banks. *Communications of the ACM* (Association for Computing Machinery) 13 (6): 377–387.

Dodds, W.K., and R.M. Oakes. 2008. Headwater influences on downstream water quality. *Environmental Management* 41(3), 367-377.

ESRI (Environmental Systems Research Institute), 2010. (cit. 2010-04-01) http://webhelp.esri.com/arcgisdesktop/9.3/index.cfm?TopicName=welcome

Gleyzer, A., M.Denisyuk, A. Rimmer, and Y. Salingar. 2004. A fast recursive GIS algorithm for computing Strahler stream order in braided and nonbraided networks. *Journal of the American Water Resources Association* 40 (4), 937–946.

Hornby, D. 2010. RivEX – A vector network processing tool for ArcGIS 9 [online]. D. Hornby. Available from: http://www.rivex.co.uk (Accessed 15 April 2010).

Johnson, B.L., W.B. Richardson, and T.J. Naimo. 1995. Past, present, and future concepts in large river ecology. *Bioscience* 45(3), 134-141.

Jaques, K. 2007. Base features derived watershed delineation and hydrocoding - Strahler order coding version 1.0. August 2007. Department of Sustainable Natural Resources, Alberta, Edmonton, Canada.

Miller, S.N., P. Guertin, and D.C. Goodrich.1996. Linking GIS and Geomorphologic Field Research at Walnut Gulch Experimental Watershed. American Water Resource Association. Pages 40-44, in *Symposium on GIS and Water Resources.* Sept 22-26, 1996. Ft. Lauderdale, FL.

Mourier, B., C. Walter, and P. Merot. 2008. Soil distribution in valleys according to stream order. *Catena* 72 (3), 395-404.

Prepas, E.E. Burke, Janice M. MacDonald, J. Douglas, G. Putz, and D.W. Smith 2008. The FORWARD Project: objectives, framework and initial integration into a detailed forest management plan in Alberta. *Forestry Chronicle* 84 (3), 330-337

Rosgen, L. and H.L. Silvey. 1996. *Applied River Morphology*. Wildland Hydrology Books, Fort Collins, CO.

Strahler, A.N. 1952. Dynamic basis of geomorphology. *Geological Society of America Bulletin* 63, 923-938.

Strahler, A.N. 1957. Quantitative analysis of watershed geomorphology. *Transactions of the American Geophysical Union*. 38(6), 913–920.

Safe Software. 2010. Stream Order Calculator [online]. http://www.fmepedia.com/index.php/StreamOrderCalculator (Accessed 15 April 2010)

Whitaker, S., L. Stanislawski and M. Hamann. 2002. Automated stream leveling for the high-resolution National Hydrography Dataset Pages 121-125, in Proceedings of the 22nd Annual ESRI International User Conference, July 8–12, 2002. San Diego. CA.

Vannote, R.L., G.W. Minshall, K.W. Cummins, J.R. Sedell, and C.E. Cushing. 1980. The river continuum concept. Canadian Journal of Fisheries and Aquatic Sciences 37:130-137.

Appendix A – SQL Code

```
use strahler
go
alter table segments add curr_ord int
alter table segments add braid_lvl int
go
create view coded_segments as
select * from segments
where code > 0
go
create view uncoded_segments as
select * from segments
where code = 0
go
create view end_nodes as
select node from nodes
inner join segments
on nodes.node = segments.to_node
where segments.code = 0 or segments.code = segments.curr_ord
go
create view start_nodes as
select node from nodes
inner join segments
on nodes.node = segments.from_node
where segments.code = 0 or segments.code = segments.curr_ord
go
create view end_node_frequency as
select node, count(node) as
frequency from end_nodes group by node
go
create view start_node_frequency as
select node, count(node) as
frequency from start_nodes group by node
go
create view bridge_nodes as
select * from end_node_frequency
inner join segments on
end_node_frequency.node = segments.to_node
where end_node_frequency.frequency = 1 and
segments.code = segments.curr_ord
go
create view braid_nodes as
select * from start_node_frequency
inner join segments on
start_node_frequency.node = segments.from_node
where start_node_frequency.frequency > 1
go
update segments set curr_ord = 0
update segments set braid_lvl = 0
go
update segments set braid_lvl = 1
where segments.from_node in (
select from_node from segments
inner join start_node_frequency on
segments.from_node = start_node_frequency.node
where start_node_frequency.frequency > 1 )
```

```
go

while (select min(code) from segments) = 0
begin
  update segments set curr_ord = curr_ord + 1
  update uncoded_segments set code = curr_ord
  where segment in (
  select segment from uncoded_segments
  where from_node not in
  (select node from nodes
  inner join uncoded_segments
  on nodes.node = uncoded_segments.to_node))
  while (select count(uncoded_segments.segment)
    from uncoded_segments
    inner join bridge_nodes on
    uncoded_segments.from_node = bridge_nodes.node) > 0
  begin
    update uncoded_segments set code = curr_ord
    where uncoded_segments.segment in (
    select uncoded_segments.segment from uncoded_segments
    inner join bridge_nodes on
    uncoded_segments.from_node = bridge_nodes.node )
    update coded_segments set braid_lvl = 1
    where coded_segments.segment in (
    select coded_segments.segment from coded_segments
    inner join bridge_nodes on
    coded_segments.from_node = bridge_nodes.node
    where bridge_nodes.node in (
    select bridge_nodes.node from bridge_nodes
    inner join coded_segments on
    bridge_nodes.node = coded_segments.to_node
    where coded_segments.braid_lvl = 1 ) )
  end

  update uncoded_segments set code = curr_ord
  where uncoded_segments.from_node in (
  select uncoded_segments.from_node from uncoded_segments
  inner join coded_segments on
  uncoded_segments.from_node = coded_segments.to_node
  where coded_segments.braid_lvl > 0 )

  while (select count(uncoded_segments.segment)
    from uncoded_segments
    inner join bridge_nodes on
    uncoded_segments.from_node = bridge_nodes.node) > 0
  begin
    update uncoded_segments set code = curr_ord
    where uncoded_segments.segment in (
    select uncoded_segments.segment from uncoded_segments
    inner join bridge_nodes on
    uncoded_segments.from_node = bridge_nodes.node )
  end
end
```

Appendix B-1 – Creating a Topologically Structured Network

None of ESRI's native data formats that are editable in ArcGIS have an implicit topological data model. For example, neither a line's start/end nodes nor its left and right neighboring polygons are implicitly stored in the data model. The direction of a line segment is implicitly defined in the data structure from source to destination. But the data model won't reveal whether the connecting node between two lines is the start or end node of a line segment. However, a fully topologically structured stream network data model is required for processing with the SQL scripts described in this paper. For that reason, we report a method on how to create a topological data model in ArcMap with the Python scripts included in Appendix B-2. The following description outlines the process of creating a dataset of nodes and identification and removal of redundant nodes.

First, we create nodes at the beginning and end of each line segment signifying them as either *Start* or *End* node, respectively, while also providing a unique ID-value for each node. At the same time, the ID of its source line segment is also stored with each node, which is significant in the final step. Also, we separate *Start* and *End* nodes into two data sets: the **start** and **end** node data sets. At this state, there are many redundant node ID's because the end nodes of almost all line segments correspond exactly to the start nodes of the downstream line segments. In the next paragraph, we describe how to remove the redundant nodes, including how to deal with nodes where braiding or confluences occur in stream networks (Fig 1).

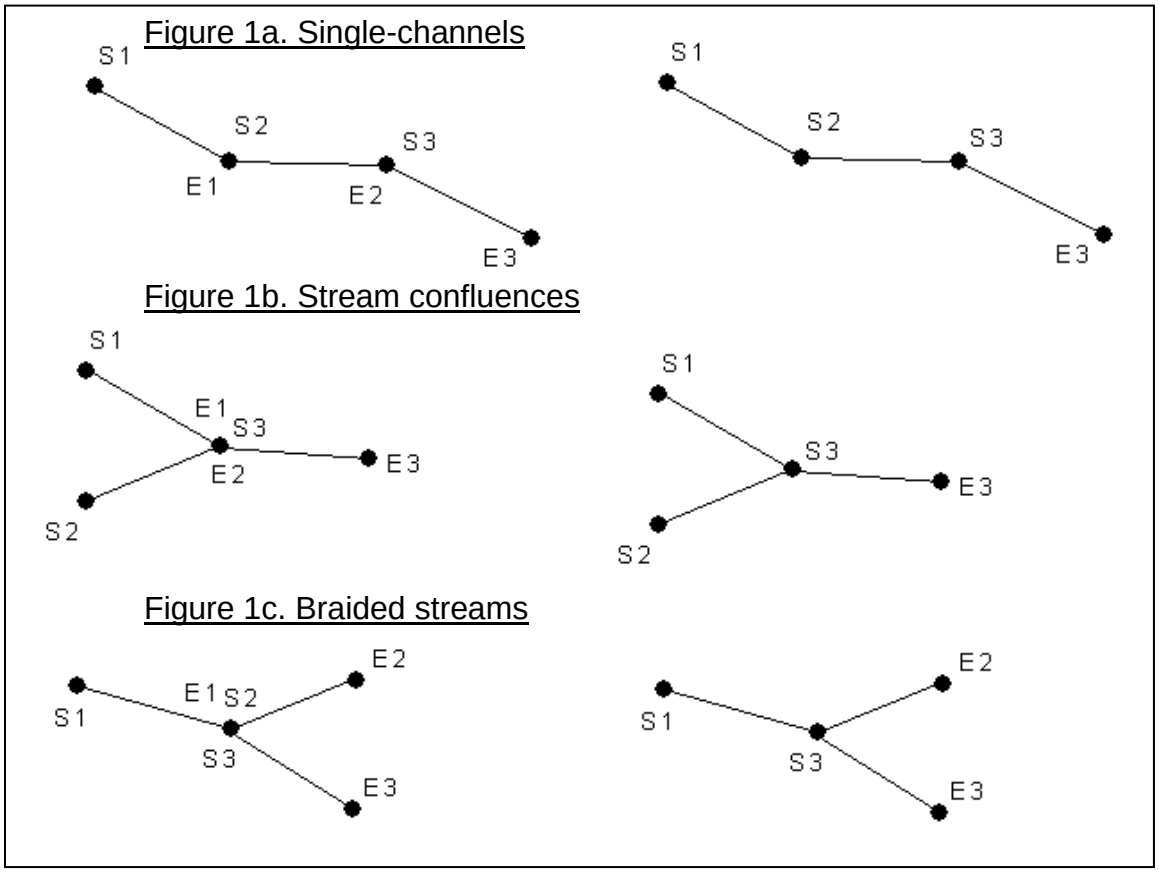

Figure 1. Node ID assignment to remove redundancy treatment in three cases

We remove redundant nodes by performing a spatial join between the **start** and **end** node data sets. *Start* and *End* nodes of adjacent upstream and downstream line segments must coincide. In single-channels the spatial join propagates the ID-value of each downstream *Start* node to the upstream **end** node data set (Fig 1a). At stream confluences (Fig 1b) the *end* nodes of tributary streams become redundant and are replaced by the *start* node ID of the downstream line segment. At braids (Fig 1c) the *start* nodes of the multiple downstream segments become redundant, and replaced by the *end* node ID of the upstream line segment.

Of course, terminal nodes representing sinks in the stream network have no corresponding *Start* nodes, and neither do nodes representing network sources, which have no corresponding *End* nodes. In those cases, attribute values of the spatially joined attributes are “0”; inadvertent breaks in the stream network are also causing “0”-values which are ideal for **quality control** of the stream network.

Finally, both **start** and **end** node data sets can be joined to the line segment table to build the fully topologically structured data model where every line segment has *Start* and *End* nodes with ID-values that match those of the respective upstream and downstream line segments. The *Start* and *End* node ID-values are propagated to the line segment table through relates. And all redundant nodes are removed through relates with the line segment table.

Appendix B-2 – Python Code

B-2.1 : mk_nodes.py

```
# Import ArcGIS Python module
import arcpy

# Variables for stream and node data sets and current directory
Dir = "C:/Gido/Publications/Strahler Order Coding/Braided"
Streams = "Streams.shp"
Nodes = "Nodes.shp"

# Set current directory
arcpy.env.workspace = Dir

# Creating node data set from segments data set
arcpy.CreateFeatureclass_management(Dir, Nodes, "Point")
arcpy.management.AddField(Dir + "/" + Nodes, "Type", "Text", "", 8)
arcpy.management.AddField(Dir + "/" + Nodes, "Segment", "Text", "", 8)

# Database cursor for looping over stream segments
StreamRows = arcpy.SearchCursor(Dir + "/" + Streams)
# Database cursor for creating (inserting) nodes
NodeRows   = arcpy.InsertCursor(Dir + "/" + Nodes)
# Point object for node creation
PointObject = arcpy.Point()

# Looping through the stream segment data set,
# starting with the first segment
Row = StreamRows.next()
while Row:
    # Get segment's spatial characteristics, and its name
    Feature = Row.getValue("Shape")
    Segment = Row.getValue("Segment")
    # Ensure to get start node even if segment is a multipart feature
    Parts = Feature.partCount
    FirstPart = Feature.getPart(0)
    FirstPoint = FirstPart.next()
    # Ensure to get the very end node
    #even if segment is a multipart feature
    LastPart = Feature.getPart(Parts - 1)
    LastPartVertexCount = LastPart.count
    # Looping through vertices of last part to get last vertex
    LastPartNextVertex = LastPart.next()
    while LastPartNextVertex:
        LastPartLastPoint = LastPartNextVertex
        LastPartNextVertex = LastPart.next()

    # Creating node from segment's start point
    newRow = NodeRows.newRow()
    PointObject.X = FirstPoint.X
    PointObject.Y = FirstPoint.Y
    newRow.setValue("Shape", PointObject)
    newRow.setValue("Segment", Segment)
    newRow.setValue("Type", "Start")
    NodeRows.insertRow(newRow)
```

```
    # Creating node from segment's end point
    newRow = NodeRows.newRow()
    PointObject.X = LastPartLastPoint.X
    PointObject.Y = LastPartLastPoint.Y
    newRow.setValue("Shape", PointObject)
    newRow.setValue("Segment", Segment)
    newRow.setValue("Type", "End")
    NodeRows.insertRow(newRow)

    Row = StreamRows.next()

# Ccalclating unique ID values for each node
Expr = "!FID! + 1"
arcpy.management.CalculateField(Nodes, "ID", Expr, "Python")

# Cleaning up variable names
del StreamRows, NodeRows
```

B-2.2 : mk_start_end_nodes.py

```
# Import ArcGIS Python module
import arcpy

# Variables for node data sets and current directory
Nodes = "Nodes.shp"
StartNodes = "Start_nodes.shp"
EndNodes = "End_nodes.shp"
Dir = "C:/Gido/Publications/Strahler Order Coding/Braided"

# Creating start node data set
arcpy.management.MakeFeatureLayer(Nodes, "Start_nodes")
arcpy.management.SelectLayerByAttribute("Start_nodes", "NEW_SELECTION",
" \"type\" = 'Start' ")
arcpy.management.CopyFeatures("Start_nodes", Dir + "/" + StartNodes)

# Creating end node data set
arcpy.management.MakeFeatureLayer(Nodes, "End_nodes")
arcpy.management.SelectLayerByAttribute("End_nodes", "NEW_SELECTION", "
\"type\" = 'End' ")
arcpy.management.CopyFeatures("End_nodes", Dir + "/" + EndNodes)

# Re-name fields names
arcpy.management.AddField(Dir + "/" + StartNodes, "Start_ID", "Long",
5)
arcpy.management.AddField(Dir + "/" + StartNodes, "Start_Seg", "Text",
5)
arcpy.management.AddField(Dir + "/" + EndNodes, "End_ID", "Long", 5)
arcpy.management.AddField(Dir + "/" + EndNodes, "End_Seg", "Text", 5)

arcpy.management.CalculateField(Dir + "/" + StartNodes, "Start_ID",
"!id!", "Python")
arcpy.management.CalculateField(Dir + "/" + StartNodes, "Start_Seg",
"!segment!", "Python")
arcpy.management.CalculateField(Dir + "/" + EndNodes, "End_ID", "!id!",
"Python")
```

```
arcpy.management.CalculateField(Dir + "/" + EndNodes, "End_Seg",
"!segment!", "Python")

arcpy.management.DeleteField(Dir + "/" + StartNodes, ["ID", "Type",
"Segment"])
arcpy.management.DeleteField(Dir + "/" + EndNodes, ["ID", "Type",
"Segment"])
```

B-2.3 : mk_spatial_joins.py

```
# Import ArcGIS Python module
import arcpy, sys

# Variables for node data sets and current directory
StartNodes = "Start_nodes.shp"
EndNodes = "End_nodes.shp"
StartEndNodes = "Start_end_nodes.shp"
EndStartNodes = "End_start_nodes.shp"
StartEndNodesStats = "Start_end_nodes_stats.dbf"
EndStartNodesStats = "End_start_nodes_stats.dbf"
Dir = "C:/Gido/Publications/Strahler Order Coding/Braided"
StatsFields =        [["End_ID", "Count"]]
CaseField =          "End_ID"

# Create spatial joins between start and end and visa versa
arcpy.analysis.SpatialJoin(StartNodes, EndNodes, Dir + "/" +
StartEndNodes, "JOIN_ONE_TO_MANY")
arcpy.analysis.SpatialJoin(EndNodes, StartNodes, Dir + "/" +
EndStartNodes, "JOIN_ONE_TO_MANY")

# Renumber end nodes to start nodes from down-stream segments
# but not for braiding downstream segments
arcpy.analysis.Statistics(EndStartNodes, Dir + "/" +
EndStartNodesStats, StatsFields, CaseField)
arcpy.management.MakeTableView(EndStartNodesStats,
"EndStartNodesStatsView")
Expr = " \"Frequency\" > 1 "
arcpy.management.SelectLayerByAttribute("EndStartNodesStatsView",
"NEW_SELECTION", Expr)
arcpy.management.DeleteRows("EndStartNodesStatsView")
arcpy.management.JoinField(EndStartNodes, CaseField,
EndStartNodesStats, CaseField)
arcpy.management.MakeFeatureLayer(EndStartNodes, "EndStartNodesView")
arcpy.management.SelectLayerByAttribute("EndStartNodesView",
"NEW_SELECTION", " \"Frequency\" > 0 and \"Start_ID\" > 0 ")
arcpy.management.CalculateField("EndStartNodesView", "End_id",
"!Start_id!", "Python")

# Renumber start nodes to end nodes from up-stream segments when
braiding
# Expr keeps only end nodes at stream braiding
arcpy.Statistics_analysis(StartEndNodes, Dir + "/" +
StartEndNodesStats, StatsFields, CaseField)
arcpy.management.MakeTableView(StartEndNodesStats,
"StartEndNodesStatsView")
Expr = " \"Frequency\" = 1 or \"End_id\" = 0 "
```

```
arcpy.management.SelectLayerByAttribute("StartEndNodesStatsView",
"NEW_SELECTION", Expr)
arcpy.management.DeleteRows("StartEndNodesStatsView")
arcpy.management.JoinField(StartEndNodes, CaseField,
StartEndNodesStats, CaseField)
arcpy.management.MakeFeatureLayer(StartEndNodes, "StartEndNodesView")
arcpy.management.SelectLayerByAttribute("StartEndNodesView",
"NEW_SELECTION", " \"Frequency\" > 0 ")
arcpy.management.CalculateField("StartEndNodesView", "Start_id",
"!End_id!", "Python")
```

B-2.4 : mk_segment_table.py

```
# Import ArcGIS Python module
import arcpy

# Variables for node data sets and current directory
Dir = "C:/Gido/Publications/Strahler Order Coding/Braided"
StartEndNodes = "Start_end_nodes.shp"
EndStartNodes = "End_start_nodes.shp"
Streams =       "Streams.shp"
Nodes =         "Nodes.shp"

arcpy.management.JoinField(Streams, "Segment", StartEndNodes,
"Start_seg", "Start_ID")
arcpy.management.JoinField(Streams, "Segment", EndStartNodes,
"End_seg", "End_ID")

arcpy.management.JoinField(Nodes, "Id", StartEndNodes, "Start_ID",
"Start_ID")
arcpy.management.JoinField(Nodes, "Id", EndStartNodes, "End_ID",
"End_ID")

arcpy.management.MakeFeatureLayer(Nodes, "NodeLayer")
Expr = " \"Start_ID\" = 0 and \"End_ID\" = 0 "
arcpy.management.SelectLayerByAttribute("NodeLayer", "NEW_SELECTION",
Expr)
arcpy.management.DeleteRows("NodeLayer")
```