Saskatchewan Environment, Stream Network Project
Creating a Seamless Water Course Network
How we did it

Process, Programming Development and Documention by: Will Stafford
Post Process written by: Dana Jones

November 2003

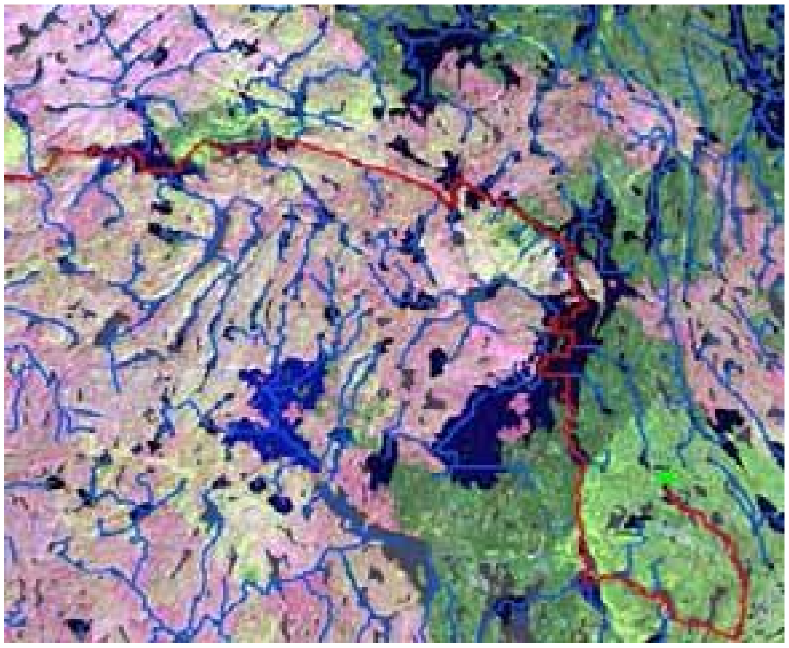


Table of Contents

Preface

This project was initiated by Saskatchewan Environment, Fish and Wildlife Branch. It was needed to utilize the Nature Conservancy Stream Classification Tools (NCSCT). The tool set gives the user the ability to create an aquatic classification system based on biotic and abiotic data. Subsequently, several other departments and organizations have shown interest in the development of the core stream network and are willing to share resources to facilitate its completion, hence this document.
The first step in the process is to create a seamless stream network. The only provincial data set available with sufficient detail is Natural Resources Canada, National Topographic System 1:50,000 water courses and lakes.

> Note: As of April 2007 all layers of Natural Resources Canada, National Topographic Data Coast to Coast to Coast is Free and Publicly available.
> It can be downloaded by NTS 1:50,000 tile as a zipped shapefile at:
> http://geogratis.cgdi.gc.ca/geogratis/en/download/topographic.html

The first hurdle to overcome was the development of through lake arc segments that connected the input / output streams. This document will focus on this last component and the logic used in connecting orphaned stream segments where applicable. The processes outlined here can be used to develop similar stream networks anywhere as long as four conditions are met. One, the stream data must exist as lines, and waterbodies as polygons, two the data must be in ArcInfo 7.x coverage format, three both data sets are in the same workspace, and four have access to ArcInfo 7+ with the grid module.
Environmental Systems Research Institute's (ESRI), ArcInfo software was used to create the core stream network. ArcEdit (a component of ArcInfo) was used to do all the editing on the network while Tom Fitzhugh's (Nature Conservancy) ArcView Stream Network Tools (http://www.freshwaters.org/info/large.shtml) were used to identity unconnected and multi-path (divergent) stream segments. In this work all multi-path segments were removed leaving only main path arcs. The final editing process used Landsat 7TM Colour Composites as background to visualise connections between lakes and streams that were not present in the original NTS50 data.

The complete Saskatchewan Stream Network and Nature Conservancy Stream Classification project results are publicly available at:
http://gisweb1.serm.gov.sk.ca/mapserver/ssn/downloads/version2/master_ssn.htm

Process for Developing the Network

General Information

Two data sets are required to create a continuous stream network. The first is the NTS 1:50,000 stream arcs that goes between water bodies, secondly, the NTS 1:50,000 water bodies, Figure 1. Both data sets can be classified into different water types. Specifically, waterbodies can be classified as, permanent, intermittent, flooded, irrigation canal and generic/unknown while streams are classified as only two types, permanent and generic/unknown. For Saskatchewan Environment the stream segments and lake types were considered the same, hence the ATC field was set to one value. The ATC classification for the NTS50 data sets and appears as the following:

Stream Arcs

ATC	Code
1463	Generic/Unknown

Waterbody Polygons

ATC	Code
0	Island
1450	Intermittent Water
1452	Permanent Water
1454	Flooded Area
1453	Irrigation Canal
1449	Generic/Unknown

It is not imperative that the starting data sets be categorized in the above fashion, but the field ATC must be there and it must be populated with 1463 for all stream arcs, 1452 for all water bodies and 0 for all islands for the process to be most effective.

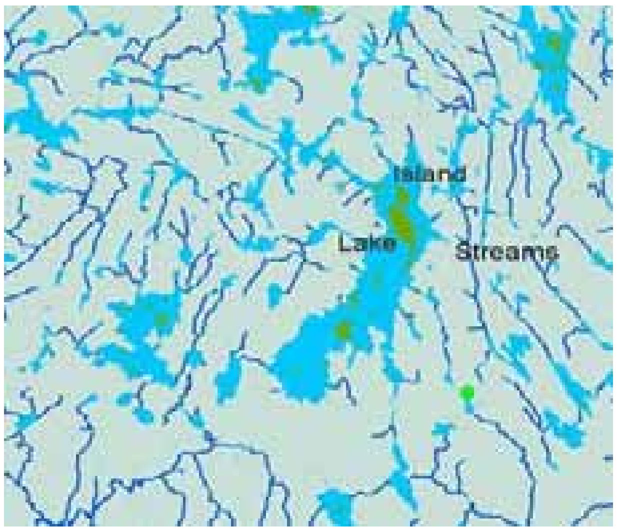


Figure 1: National Topographic Series 1:50,000 waterbody and watercourse.

Developing the Starting Data Sets

Saskatchewan Environment chose the watershed as the logical unit for processing. Several watershed boundaries have been developed for Saskatchewan but we chose to use the Prairie Farm Rehabilitation Association (PFRA) watershed boundaries Version 3. It contains 33 major watersheds that cover Saskatchewan. From a computational stand point the Churchill River above junction with Reindeer River, Churchill River below the junction with Reindeer River and Athabasca River below Athabasca were subdivided into small units because of their large geographic extent, Figure 2. Those small areas were processed then recombined at the end. One watershed is comprised of several entire watershed, Figure 3. Once appended the watershed boundary were used to select out all streams and lakes that fell inside it. The appended and selected data for streams was called water_cl and water_ba for the waterbodies. Those two compiled data sets are the starting data on which the stream processing amls run.

Developing the Initial Stream Skeleton

The development of the initial skeleton evolved over the project. Conceptually, the streams and lakes were converted to raster data then thinned down to single lines of raster data which were then converted into vector data. The following steps were used to create the base stream skeleton. A conceptual description is given followed by the explicit ArcInfo commands in courier italics.

Stream Network Processing amls

Create_thin_lines_(watershed_name).aml

The first aml that runs is create_thin_lines_(watershed_name).aml. If stream networks are required for multiple watersheds than this aml needs to be edited for each watershed. Below is a section of code for creating a fishnet data set that is based on the extent of each watershed and those values need to be modified by the user for this aml to function properly.

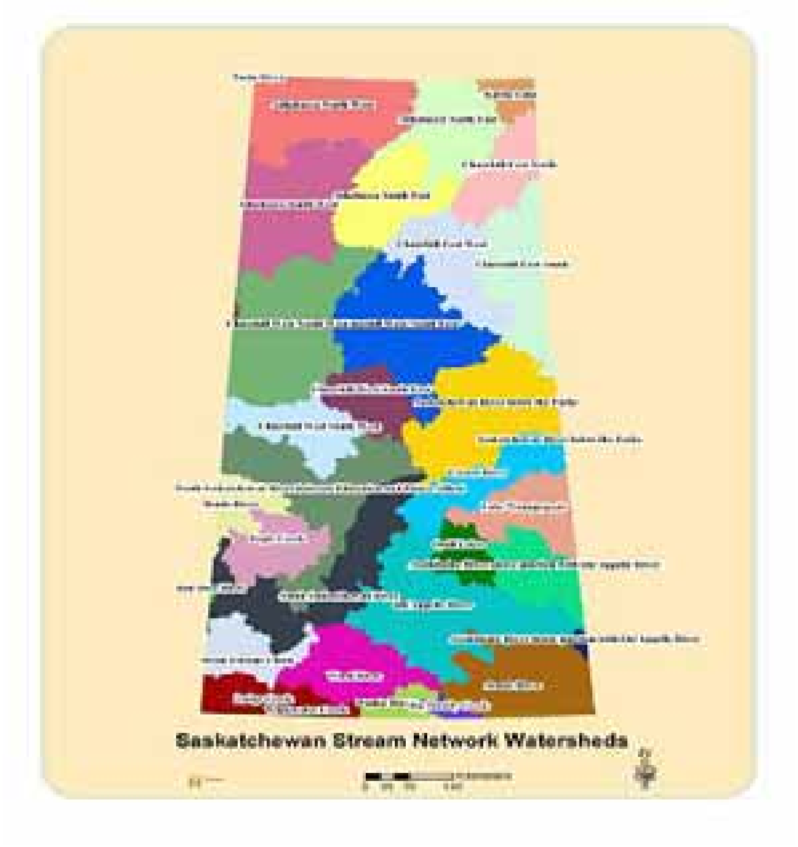


Figure 2: Watershed areas used for processing.

Start of create_thin_line_churchwest_northwest.aml
Set precision to double double to get the highest accuracy possible. Older data sets with low precision values must have precisions set to double double.

```
/* 
create_thin_line_churchwest_
northwest.aml
precision double double
```

Buffer the stream network arcs with round ends to act as a clipping cover to select all lakes that touch streams. Ten metres were used because trial and error illustrated that it was the best value to ensure all lakes that touched stream arcs were included. It is possible to run this aml with smaller or large grid cell sizes. However if modifying the default 10m cell size to another value it should be based on the size of the watershed. Smaller watersheds can have smaller cell sizes. When cell size is modified the spatial location of arcs at the interface between lake arcs and stream arcs will move proportional to the chosen cell size at the lakes edges. For the Saskatchewan work, decisions were made that 10m either way was well within the tolerance of the starting data. The most important area to watch is the spatial representation of in lake arcs in double line water course that are meandering. In very meandering instances the skeleton may snap across a large loop stream section, because the loop is smaller than 2x the cell size. In Saskatchewan double line water course may form a very tight loops which are less than 20m apart. The resulting stream skeleton lines will jump across the loop, versus going around the loop in the double line water course. If the operating cell size is modified from 10m, one needs to watch what happens to meandering double line water courses and modify all of commands that create grids of 10m.

The process starts with selecting out lakes that touch streams, Figure 4.

```
buffer water_cl strm_buff # # 10 0.001 line round ful
clip water_ba strm_buff lake_clp poly 0.001
build lake_clp poly
```

Develop items for relating clipped lake data from buffers to original lakes to identify lakes that touch streams.

```
copy water_ba lake_2
build lake_2 poly
additem lake_2.pat lake_2.pat lake-id 4 5 b
additem lake_clp.pat lake_clp.pat lake-id 4
5 b
tables
sel lake_2.pat
calc lake-id = lake_2-id
sel lake_clp.pat
calc lake-id = lake_clp-id
quit
joinitem lake_2.pat lake_clp.pat lake_2.pat
lake-id # linear
```

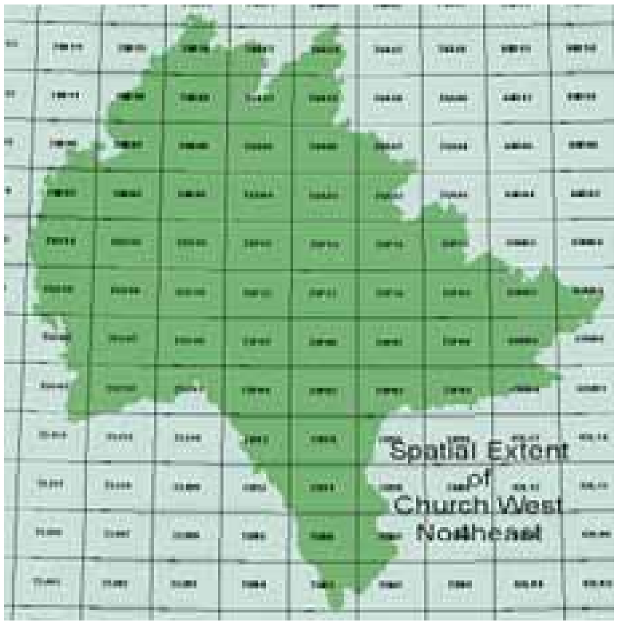


Figure 3: Extent of Church West Northeast Watershed.

Save out lakes that touch stream arcs to a new data sets called sel_lakes which contains all lakes that touch a stream.

```
editcov lake_2
editf poly
sel all
reselect lake_clp-id > 0
put sel_lakes
quit
build sel_lakes poly
```

From the selected lakes coverage copy to a new coverage and dissolve away all boundaries between lakes for lakes that cross NTS50 sheet boundaries so that lakes become whole, Figure 5.

```
copy sel_lakes tmp_lakes
build tmp_lakes poly
Create an islands coverage for later use in
costpath analysis.
arcedit
editcov water_ba
editf poly
asel all
reselect atc = 0
put islands
quit
build islands poly
```

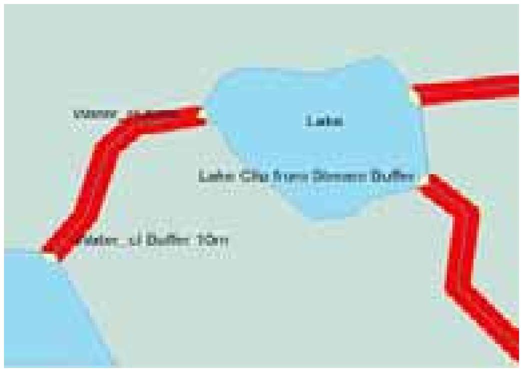


Figure 4: Selecting out lakes with streams.

Dissolve away islands inside lakes to create the boundary

edges for the costpath analysis in the cost_path.aml and the gridthin functions in this aml.

```
tables
sel tmp_lakes.pat
reselect area > 0
calc atc = 1450
quit
dissolve tmp_lakes sel_lake_dis atc poly
```

Make sure all polygons have label points. Those label points are used as start points for costpath analysis in the cost_path aml.

```
createlabels sel_lake_dis
build sel_lake_dis poly
build sel_lake_dis line
```

Experience has taught that gridline has problems thinning down large lakes to single arc segments. To avoid those errors, create holes (voids) in lakes larger than 50,000 m2 over the entire lake in a regular pattern which helps gridline do its job. The first step is to select the lakes involved in this process.

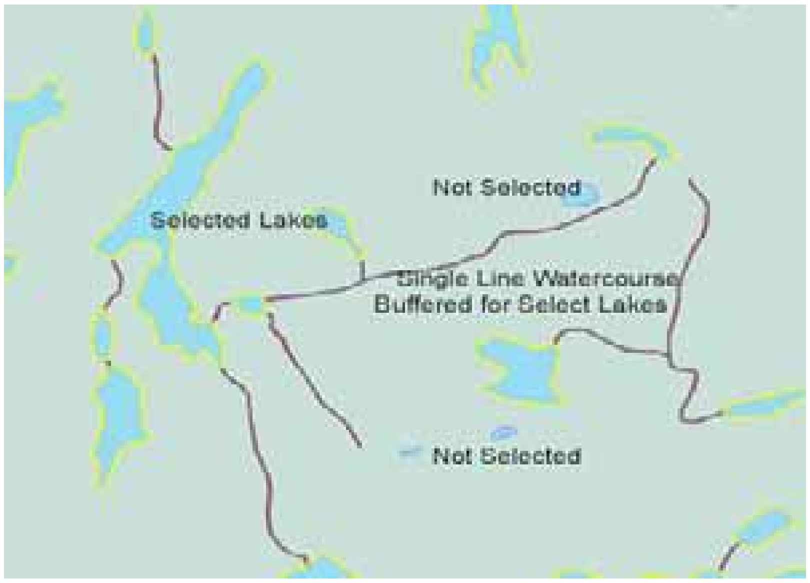


Figure 5: Selected lakes based on watercourse buffer.

```
arcedit
editcov sel_lake_dis
editf poly
asel all
reselect area > 50000
put sel_lake_net
quit
build sel_lake_net poly
buffer sel_lake_net net_lake_ins # # -100 0.001 poly # #
build net_lake_ins poly
```

This section describe the process used to create a data set that will put holes in the large lakes grid, those holes facilitate gridline and reduces the number of problems when using gridline. This process involves the use of an arc fishnet coverage converted to points at arc intersections. User input is required to calculate the extent of the starting fishnet. The user must modify the values in the create_thin_line_(watershed_name).aml.
The following approach was used. Open the watershed polygon data in ArcView, and using the coordinate values that appear in the upper right corner of ArcView's window, develop the numbers for the fishnet. First, place the mouse cursor at the lower left corner and ensure that the cursor is OUTSIDE the most southern and western values of the polygon representing the watershed of interest. Record the values shown as the fishnet coordinate, see below. Then place the cursor at the most northerly extent of the

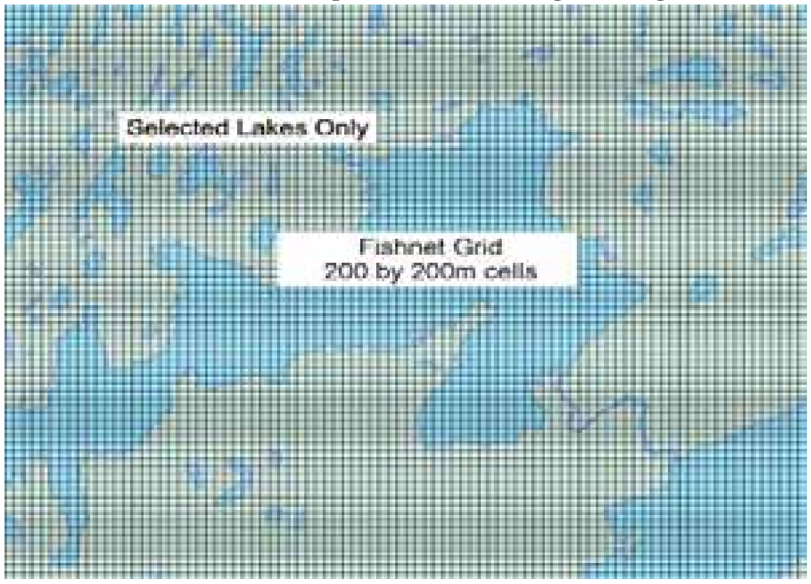


Figure 6: Results from fishnet command.

watershed boundary and record the upper Y value. Lastly, move the cursor all the way to the easterly limit of the watershed boundary and record the most easterly x value. The four required parameters to complete the section below are defined. Required values for Generate to create a fishnet are obtained from ESRI's ArcInfo Documentation.

```
generate net
fishnet coordinate (x,y)Lower left corner value.
y-axis origin (x,y) Upper left corner vertically above the lower left corner.
cell size (width,height) Fishnet cell size (width metres, height metres).
number of rows,columns Number of rows, number of columns the fish net is.
```

To find fishnet coordinate (x,y) use;
The lower left X,Y values you recorded
To find y-axis origin (x,y) use;
The maximum y value obtained and the same x value from the lower left X,Y values you recorded
To find cell size (width, height) use;
200m or a different cell size, 200 by 200m worked best for us.
To find number of rows, columns use;
The positive value of (maximum Y - minimum Y) / cellsize 200m for rows.
The positive value of (maximum X - minimum X) / cellsize 200m for columns.

Below is an example fishnet for the Church East South Watershed which is then built as a point data set. Record values from ArcView coordinate box when viewing the Church East South watershed;
Lower x,y or most westerly and southerly values = 334200m, 5983000m.
maximum y or north value = 6329000m.
minimum x or east value = 45400m.
Those values were input to the generate command below, Figure 6;

```
generate net
fishnet
334200,5983000 Notice the two X values are same for fishnet coordinate and Y-axis
origin
334200,6329000
200,200
1730,1444    1730 = (6329000 - 5983000) / 200  and 1444 = (334200 - 45400) / 200
quit
build net point
```

This next code takes the fishnet points and clips them inside the lake boundaries, buffers the clipped points by 30 metres, Figure 7 and adds an item, which will be used to create a grid data set based on the values of 1 or 0.

```
clip net net_lake_ins net_clp point #
buffer net_clp net_clp_buf # # 30 # point # #
additem net_clp_buf.pat net_clp_buf.pat cel_v 5 5 i
tables
sel net_clp_buf.pat
calc cel_v = 1
quit
```

The following code enters the ESRI's grid module. In grid, this aml converts several vector data sets to raster data sets at a resolution of 10m. 10m cell size was used because it was half of the known accuracy of 20m the starting data.

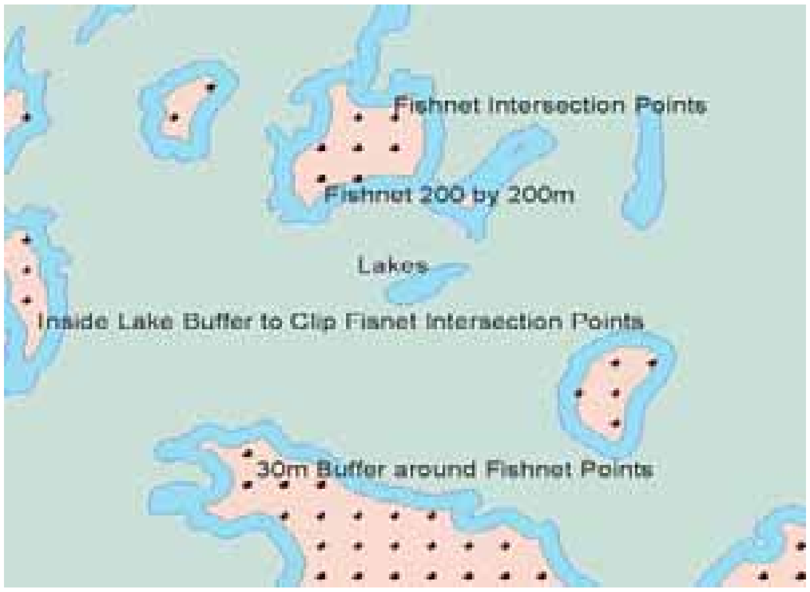
```
grid
lake_grid =
polygrid(sel_lake_dis,sel_lake_dis#,#,#,10)
```

```
stream_g =
linegrid(water_cl,water_cl#,#,#,10,nodata)
island_g = polygrid(islands, islands#,#,#,10)
net_grd = polygrid(net_clp_buf,cel_v,#,#,10)
```

To develop in lake arcs that flow around islands and stay away from the shore, nodata values for those areas are needed. Therefore, islands are burned as no data values to the lake grid. The nodata values act as barriers to the gridthin i.e. keeps the raster thinning operation contained inside of lakes and off of island areas.

```
lake_isl_grd =
con(isnul(island_g),
lake_grid,setnul(island_g)
)
```

The lake grid with nodata island is overlain on the gridded stream network and put together to form a raster data set of lakes connected to streams.



Fishnet Intersection Points
Fishnet 200 by 200m
Lakes
Inside Lake Buffer to Clip Fisnet Intersection Points
30m Buffer around Fishnet Points

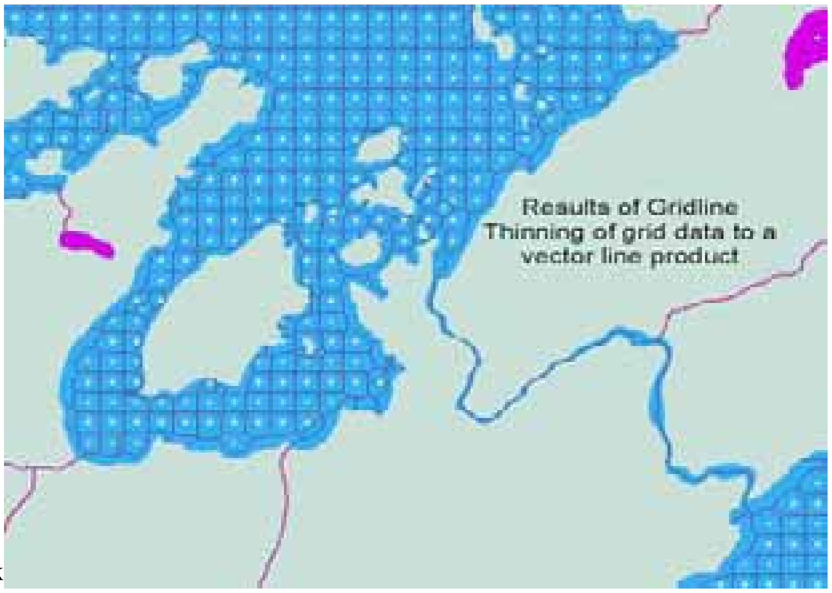
Figure 7: Holes created from buffered fishnet.

```
lake_strm_grd = con(isnul(stream_g),lake_isl_grd,stream_g)
```

Lastly, the gridded lake and stream data has more nodata holes put into it for lakes larger than 50,000 m^2. This is where the fishnet point data in it's grid format is applied, Figure 8.

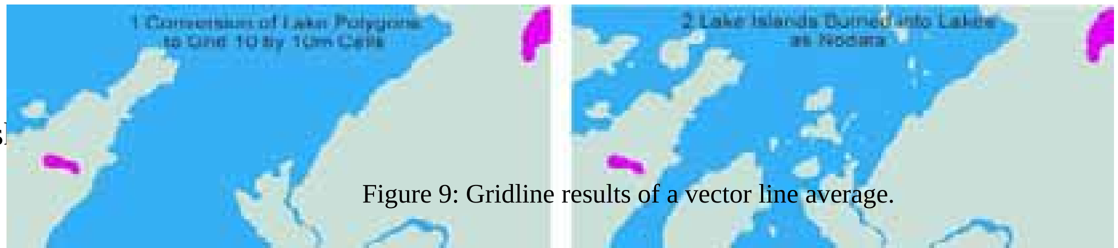
```
stream_line = con(isnul(net_grd),lake_strm_grd,setnul(net_grd))
```

The stream_line grid now needs to be thinned down. Thinned, means that pixels are eaten away from the positive grid values i.e. not nodata, down to a single continuous set of eight cardinal direction adjacent pixels. The thinned data look like a spider web of lines but they are still rasters. The next set of code MAY require user interaction if a suitable looping code block is not developed. In developing this process, it was discovered that ESRI's ArcInfo Grid



Results of Gridline Thinning of grid data to a vector line product

Figure 8: Development of lak



1 Conversion of Lake Polygons to Grid 10 by 10m Cells
2 Lake Islands Burned into Lakes as Nodata

Figure 9: Gridline results of a vector line average.

command gridline cannot deal with raster data sets that are more than 16000 pixels wide in the thinning process all of the time. It has been reported as a intermittent problem to ESRI. An error normally occurs indicating that it has found a intersection and cannot continue. To overcome this difficulty, simply cut up the stream_line data set into small parts < 16,000 pixels wide. Sometimes a simple splitting of the data set into two parts is enough for gridline to run. However, on complex edged, large lake raster data sets, the whole must be cut down to parts 6000 pixels wide in an iterative process for it to work.

First try the watershed without modification.

```
water_l = gridline(stream_line,data,thin,nofilter,round,value,#,#,#)
```

If it completes refer to Figure 9. If it does not complete, cut the stream_line grid data set in half, the two parts arc called one and two. One always being the most easterly part followed by two, three etc.
In doing this process one may complete but two fails. Indicating that two must be split into two parts called two1 and two2. Run those and two2 completes but two1 fails. Again split two1 into two data sets two11 and two12, see the following conceptual example.

stream_line 36,000 pixels wide by 38,000 pixels tall. (This is a normal sized data set for Saskatchewan)

```
water_l = gridline(stream_line,data,thin,nofilter,round,value,#,#,#),
```

FAILS
split stream_line into dataset one and dataset two.
`one_l = gridline(one,data,thin,nofilter,round,value,#,#,#),` FAILS
`two_l = gridline(two,data,thin,nofilter,round,value,#,#,#),` Passes and two_l vector data set is created.

split one into one1, one2
`one1_l = gridline(one1,data,thin,nofilter,round,value,#,#,#)`, Passes and one1_l vector data set is created.
`one2_l = gridline(one2,data,thin,nofilter,round,value,#,#,#)`, FAILS

split one2 into one21 and one22
`one21_l = gridline(one21,data,thin,nofilter,round,value,#,#,#)`, Passes and one21_l vector data set is created.
`one22_l = gridline(one22,data,thin,nofilter,round,value,#,#,#),` Passes and one22_l vector dataset is created.

All of the vector data sets are now appended together to form one water_l vector data set which is cleaned and built as a line data set.

```
append water_l line all
two_l
one1_l
one21_l
one22_l
end
```

The description above got converged into the cut and paste code below to handle those situations. This problem generally occurred on every Saskatchewan Watershed greater than 10,000 by 10,000 pixels. If a smaller watershed were chosen i.e. subdivisions of the 33 major ones, it would not of been such an issue and gridline would have been able to process it. It made logical sense to keep every thing together at that large scale of 33 major watersheds even though problems were encountered. We are unsure if using smaller watershed would have reduced the effort required for editing and verifying the data compared to the larger watersheds we used.

The first step was to create a square boundary around the whole watershed. The boundary was created using ArcView 3.2a because of its ease of use. The resulting polygon shapefile was converted to a coverage. The square boundary can also be created in ArcEdit, depending on personal preference, Figure 10.

In ArcView, add the watershed boundary.
Create a new polygon theme, View, New Theme, Polygon, Okay, Give it a name and put it in the stream network workspace.
Select the box symbol and create a square polygon, It is okay to go several kilometers larger than the watershed.
Theme Stop Editing, Save Edit Yes.

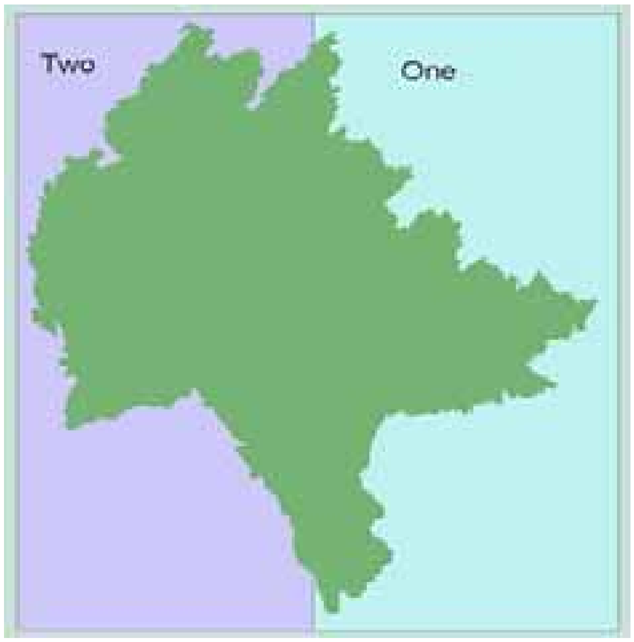


Figure 10: Creating the streamline clipping cover one and two.

From this point on all of the editing done to create those clipping coverages is done in ArcInfo's ArcEdit module after you convert the shape file into a polygon coverage.

```
shapearc square_bnd sq_bnd
build sq_bnd poly
arcedit
display 9999
editcoverage sq_bnd
editfeature arc
drawenvironment arcs
draw
intersecarcs all
add /* add a vertical line down the
middle of gsq_bnd to split it into
/* two polygons  with large dangles for ease of editing
select many /* select and delete the dangles from
the added arc
delete
build /* build the data as a polygon
editfeature  poly
select one /* select the east polygon i.e. one
put one
sel one /* select the west polygon i.e. two
put two
quit
n /* don't save changes to sq_bnd keep it as the ordinal
/*  one polygon boundary
```

Now we move into grid to clip stream_line into two data sets and try gridline to see if it will complete with only the one cut.

```
grid /* start grid
gridclip stream_line strm_one cover one
gridclip stream_line strm_two cover two
one_l = gridline(strm_one,data,thin,nofilter,round,value,#,#,#)
two_l = gridline(strm_two,data,thin,nofilter,round,value,#,#,#)
quit /* quit grid
```

If all is well i.e. gridline is able to create the two vector coverages one_l and two_l than append the two data sets to create water_l and move on to the next section.

```
append water_l line all
one_l
```

```
two_l
end
```

The following code details two more splits which create 4 data sets that might be needed for a successful conversion of the grid data to vector data, all of which need to be append together to form the final vector coverage. In Saskatchewan, the ChurchWest watershed northwest section required six parts for it to work.

This looks after the one polygon coverage.

```
arcedit
display 9999
editcoverage one
editfeature arc
drawenvironment arcs
draw
intersecarcs all
add /* add a vertical line down the middle of one to split it into
/* two polygons with large dangles for ease of editing
select many /* select and delete the dangles from the added arc
delete
build /* build the data as a polygon
editfeature  poly
select one /* select the east polygon i.e. one
put one1
sel one /* select the west polygon i.e. two
put one2
quit
build one1 poly
build one2 poly
```

This looks after the two polygon coverage.

```
arcedit
display 9999
editcoverage two
editfeature arc
drawenvironment arcs
draw
intersecarcs all
add /* add a vertical line down the middle of one to split it into
/* two polygons with large dangles for ease of editing.

select many /* select and delete the dangles from the added arc
delete
build /* build the data as a polygon
editfeature  poly
select one /* select the east polygon i.e. one
put two1
sel one /* select the west polygon i.e. two
put two2
quit
build two1 poly
build two2 poly
```

Now enter grid and start work there for one1, one2

```
grid /* start grid
gridclip stream_line strm_one1 cover one1
gridclip stream_line strm_one2 cover one2
one1_l = gridline(strm_one1,data,thin,nofilter,round,value,#,#,#)
one2_l = gridline(strm_one2,data,thin,nofilter,round,value,#,#,#)
```

```
quit /* quit grid
```

Now enter grid and start work there for two1, two2

```
grid /* start grid
gridclip stream_line strm_two1 cover two1
gridclip stream_line strm_two2 cover two2
two1_l = gridline(strm_two1,data,thin,nofilter,round,value,#,#,#)
two2_l = gridline(strm_two2,data,thin,nofilter,round,value,#,#,#)
quit /* quit grid
```

Append together the vector results to create water_l

```
append water_l line all
one1_l
one2_l
two1_l
two2_l
end
```

Lastly clean and build the water_l data set. The water_l was cleaned with a movement of 12m which is 2 metres greater than the source raster data set cell size, Figure 11. Experimentation illustrated that this was the best value to use as a compromise.

```
build water_l line
clean water_l water_l_cl 10 12
build water_l_cl line
The data set water_l_cl is the final result of
create_thin_lines_(watershed_name).aml. It is used
in the cost_path.aml.
```

Create_start_end_points.aml

Create_start_end_points.aml job is used to create the starting and ending point data sets that are used by costpath functions to develop the internal lake skeleton and ensure that the lake skeleton connects to stream entry and exit points. It accomplishes this by selecting out all arc nodes where streams enter lakes and converts those to end points, and all lake labels as start points. Start, end points are used by costpath to tell it where to start its search from across a grid surface to find the least costpath way to get to end points. It is the path between those two points that becomes the in lake skeleton. To avoid corruption of the data, work on a backup copy of water_cl.

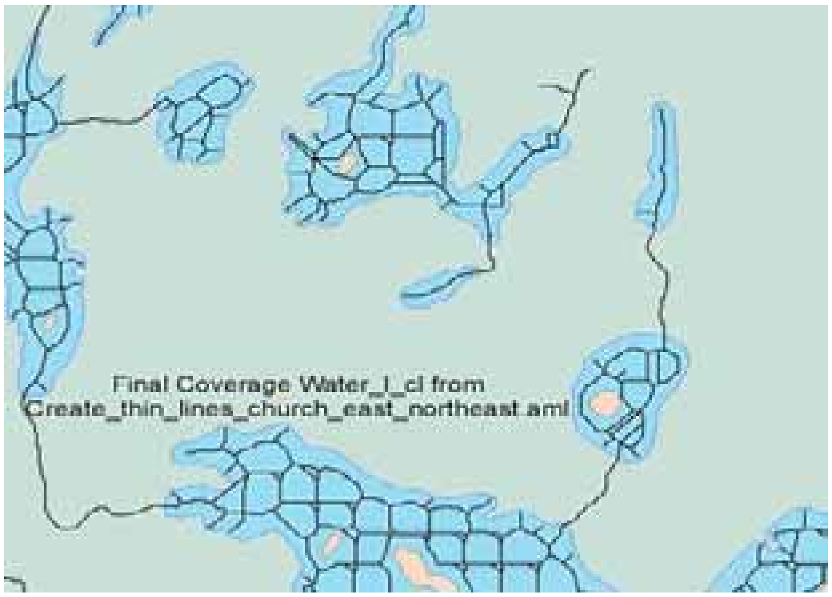

Final Coverage Water_l_cl from
Create_thin_lines_church_east_northeast.aml

Figure 11: Final result of create_thin_lines_church_west_northeast.aml

```
&do
kill water_cl_org all
&end
copy water_cl water_cl_org
```

Extra items in water_cl.aat data table are dropped since they are not required.

```
dropitem water_cl.aat water_cl.aat mapid water_cl_i elevation type accuracy ate atz
atg atc
code_gener id entityname tmp1_id order_id order
```

This aml works with two data sets sel_lake_dis and water_cl, below is code to ensure the relate item lake_bnd is clean and clear.

```
&if [iteminfo sel_lake_dis -line lake_bnd -exists] &then
    &do
    dropitem sel_lake_dis.aat sel_lake_dis.aat lake_bnd
    &end
&if [iteminfo water_cl -line lake_bnd -exists] &then
    &do
    dropitem water_cl.aat water_cl.aat lake_bnd
    &end
&if [iteminfo water_cl -line value -exists] &then
    &do
    dropitem water_cl.aat water_cl.aat value
    &end
additem sel_lake_dis.aat sel_lake_dis.aat lake_bnd 4 5 n
additem water_cl.aat water_cl.aat lake_bnd 4 5 n
```

Populate lake_bnd item in both data sets to identify which arc segments come from which dataset, sel_lake_dis or water_cl.

```
tables
asel all
calc lake_bnd = 1000
sel water_cl.aat
asel all
calc lake_bnd = 100
quit
```

The two line data sets sel_lake_dis arcs and water_cl arcs are appended as lines, cleaned and built as lines.

```
append water_arcs line all
sel_lake_dis
water_cl
end
clean water_arcs water_arcs 0.001 0.001
build water_arcs line
```

ArcEdit is used to select then unsplit all of the lake boundary arcs. This leaves nodes or intersections at locations where streams enter lakes. This process has a drawback because it places and identifies nodes at some locations where a stream does not enter or exit a lake. It only occurs on lakes with very large boundaries, because the maximum arc length is reached and a node must be added. Its those extra nodes that create issues later. They produce in-lake arc segments that run off to the side of lakes where streams do not enter or exit. This is corrected by manually selecting those arcs and deleting them during the editing process.

```
arcedit
```

```
editcov water_arcs
editf arc
sel all
unsplit
save
sel all
resel lake_bnd = 1000
put lake_bnd
quit
renode lake_bnd
build lake_bnd line
nodepoint lake_bnd lake_ends
build lake_ends point
additem lake_ends.pat lake_ends.pat lake_bnd 4 5 n
```

This section creates the starting points for costpath. The starting points are always the label points of the lake, which were selected out from the sel_lake_dis coverage and converted to a point data set, Figure 12.

```
arcedit
editcov sel_lake_dis
editf label
sel all
put lake_labels
quit
build lake_labels point
additem lake_labels.pat lake_labels.pat lake_bnd  4 5 n
tables
sel lake_labels.pat
calc lake_bnd = 1000
sel lake_ends.pat
calc lake_bnd = 100
quit
```

The last step is the creation of a point data set for all of the stream entry and exit nodes around a lake and the lake label point. This data set is used for lake elevation work later.

```
append node_elv point all
lake_labels
lake_ends
end
build node_elv point
```

Cost_path.aml

This aml takes the results of create_thin_lines_(watershed_name).aml and start_end_points.aml and brings them together to create the base stream skeleton that goes through the costpath functions to create the connections between stream entry and exit points and the lake label points. The resulting skeleton is merged with the original water_cl stream arcs and both data sets are snapped together in an

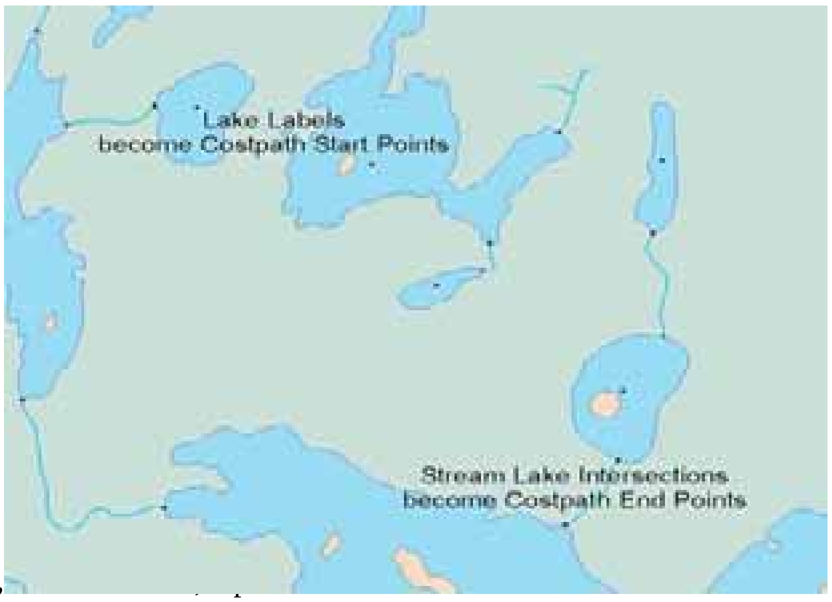

Lake Labels
become Costpath Start Points
Stream Lake Intersections
become Costpath End Points

Figure 12: Start and End Points for Costpath functions.

iterative process. This aml has one major sub routine which adds extra arcs inside larger lakes to unsure that the cost surface grid developed from those arcs will intersect with all start and end points used for costpath.
The starting data set is a version of water_l_cl clipped by the sel_lake_dis data, which is just the spider web of arcs that go through lakes of interest Figure 13.

```
&ty Creating a clipped version of the 15m cleaned water_l_cl that go through a lake
clip water_l_cl sel_lake_dis water_lclp line 0.001
build water_lclp line
```

The following code is the call to the sub routine called buffer which will add extra arcs to ensure a functional cost surface result. This component was added when the internal lake skeleton failed to develop because the start point was not covered or within 105 metres of the initial cost surface developed from water_l arcs.

```
 &call buffer
&ty Creating buffers for large lake and intersecting those with the thinned grid
lines
```

The cost_path.aml returns to this positions after the completion of the buffer routine. This next section is somewhat involved. It deals with the creation of grid data sets that are used in costpath functions to develop the internal lake skeleton. The first step is to develop an analysis mask which increase the processing speed. Some commands in this section can take several hours to complete. For the largest watersheds processing time for one specific command costpath was 14 hours. The other commands are fairly fast to complete.

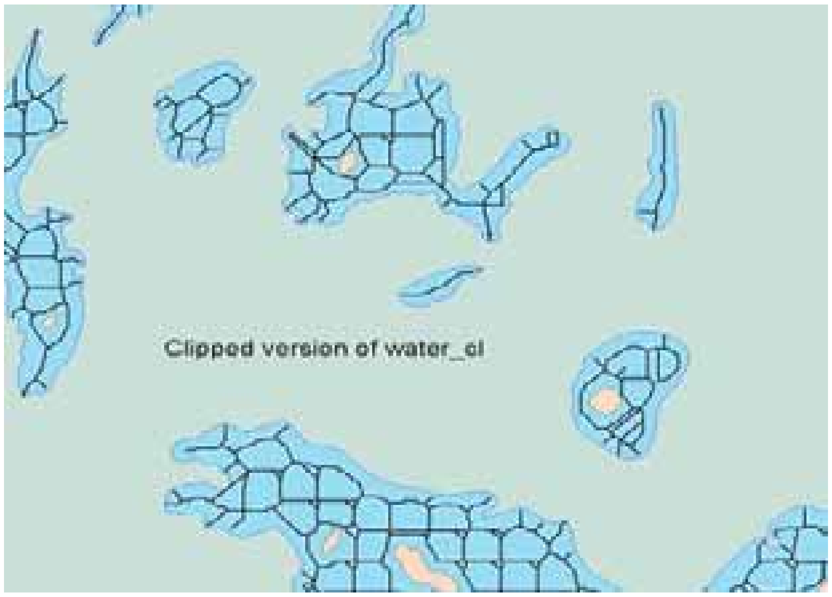


Figure 13: Clipped version of water_cl by select lakes.

```
&ty Create a mask from the
selected lakes data set that
is 20m larger than the
starting lake size.
buffer sel_lake_dis
sel_lake_buf # # 20 0.001 poly
round full
polygrid sel_lake_buf mask # # #
10
y
&ty Start grid for the analysis and creation of the lake skeleton
grid
```

This section of code has many visible comments that appear while running so the user has an idea of how it is doing.

```
&ty Step 1, need starting point for costpath analysis which
&ty is the lake label point for each of the
&ty  lakes that have streams coming from them i.e.
labels
&ty from sel_lake_dis and must be integer grid

Create the grid data of pixels that will act as starting points.

starts = pointgrid(lake_labels,lake_labels-id,#,#,10,nodata)
start_cels = int(starts)
```

Create the grid data of pixels that will act as end points i.e. places where streams enter and exit a lake Figure 14.

```
&ty Step 2, need end points for costpath analysis
&ty which are the locations where streams
&ty intersect the lake boundary from selected lakes i.e. sel_lake_dis
ends = pointgrid(lake_ends,lake_ends-id,#,#,10,nodata)
```

Drop out of grid to go back to all default settings for grid and then start grid and set the mask extent.

```
quit
&ty Step 3, get a mask set up to constrain the analysis
&ty area to the selected lakes only
```

```
grid
setmask mask
```

The following command takes the aggregated buff_watclp line data set and creates a grid from those lines that extend out 105m from each arc, Figure 15. 105m out was used because when looking at buff_watclp, one will see that is composed of many squares that are approximately 200m wide by 200m long. If each line is buffered 105m on both sides, all the internal space will be filled with a valid cell value i.e. not nodata, Figure 15.

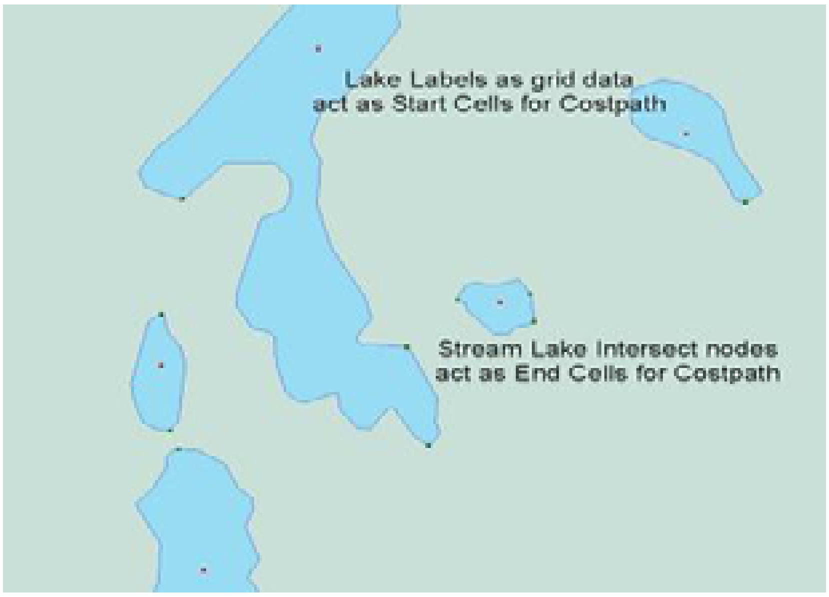

Lake Labels as grid data act as Start Cells for Costpath
Stream Lake Intersect nodes act as End Cells for Costpath

Figure 14: Start and end grid cells for costpath analysis.

```
&ty Step 4, create line
distance grid which
becomes the
&ty line_distance grid
from edge of shore to a
line
cost_grid = linedist(buff_watclp,buff_watclp,10,dist,105)
```

If the buff_watclp is large enough the 32 bit file size limit will be reached and the process will terminate. On large watersheds the grid command linedist can make this happen. You get a file seek error from the operations system (i.e. SUN Solaris 7.5). The outlined process has never been run on intel boxes with NT4 or Windows 2000 so we do not know it will happen there, but our bets are it will since both Microsoft and Sun’s OS’s are 32 bit and will suffer from the same constraints.

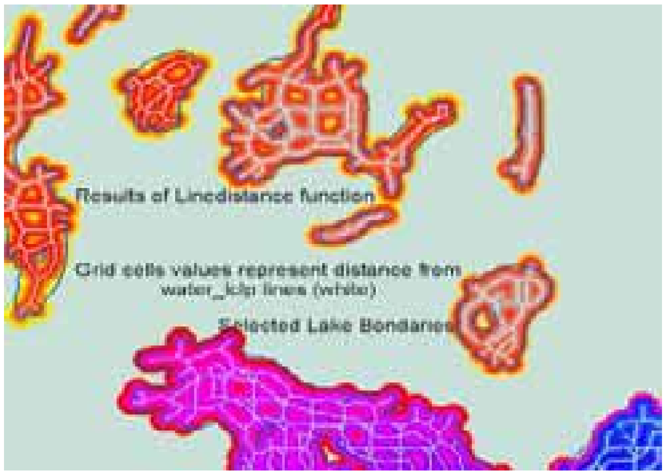

Results of Linedistance function
Grid cells values represent distance from water_kclp lines (white)
Selected Lake Bondaries

Figure 15: Line distance results.

The Solaris OS gives the following error;
FATAL ERROR
MSEEK: Invalid Argument while accessing file (seeking to position -2147483648 of -2147483648 for read): /path/to/grid/w001001.adf
Bailing out of GRID
FATAL ERROR
See ESRI article at http://support.esri.com/index.cfm?fa=knowledgebase.techarticles.articleShow&d=14574

When the above error message appears, split buff_watclp into two or more parts and run all of the costpath functions on each part. Below is the cut and paste code to do that. If the indicated problem occurs then you have most likely had a problems in create_thin_line_(watershed_name).aml. Therefore, the square box boundary has been created and at least one split has been done. If oit was not done r it was deleted, the following code for creating the split polygons and the rest of cost_path.aml commands that will develop the complete stream skeleton.

```
#######################################################
arcedit
editcoverage sq_bnd
display 9999
drawenvironment arcs
draw
editfeature arc
intersecarcs all
add /* add a vertical line down the middle of sq_bnd to split it into two polygons
with large dangles for easy of editing
select many /* select and delete the dangles from the added arc
delete
build /* build the data as a polygon
editfeature  poly
select one /* select the east polygon i.e. one
put one
sel one /* select the west polygon i.e. two
put two
quit
n /* don't save changes to sq_bnd keep it as the
ordinal one polygon boundary
build one poly
build two poly
clip buff_watclp one buff_watclp1 line 0.001
clip buff_watclp two buff_watclp2 line 0.001
grid
cost_grid1 =
linedist(buff_watclp1,buff_watclp1,10,dist,105)

cost_grid2 = linedist(buff_watclp2,buff_watclp2,10,dist,105)
setwindow cost_grid1
cost_dis1 = costdistance(start_cels,cost_grid1,back_link1,al_grid1,#,#)
setwindow cost_grid2
cost_dist2 = costdistance(start_cels,cost_grid2,back_link2,al_grid2,#,#)
setwindow cost_grid1
cost_path1 = costpath(ends,cost_grid1,back_link1,bycel)
setwindow cost_grid2
cost_path2 = costpath(ends,cost_grid2,back_link2,bycel)
setwindow cost_path1
lake_path1 = gridline(cost_path1,data,nothin,nofilter,sharp,#,#,#,15)
setwindow cost_path2
lake_path2 = gridline(cost_path2,data,nothin,nofilter,sharp,#,#,#,15)
append joined_lines line al
lake_path1
lake_path2
stream_seg
end
build joined_lines line
```

The basic stream network skeleton has been created if a split of watclp_buff was required.

```
#######################################################
```

When the ESRI grid linedist command has completed, it results in a grid data set with different cell values representing the spatial distance between its position and the source line. It is similar to the vector world buffer command. Now the cost distance function is run to develop the cost surface between a start point (lake label point) and where streams enter the lake (end points). Patience is important as this command can take time. As stated, one watershed in Saskatchewan took 1.5 days to run this command, Figure 16 and Figure 17.

```
&ty Step 5, run the cost distance calculation to develop the cost_distance
&ty from startcels i.e lake labels to end cells
cost_dist = costdistance(start_cels,cost_grid,back_link,al_grid,#,#)
```

With a completed cost surface, costpath can be run to determine the shortest distance between start points and end points for each lake, Figure 18. The output grid is comprised of a spider web of arcs that lead from each lakes label point around islands to the edge of lakes.

```
&ty Step 6, run the costpath
analysis to develop the
shortest path between
&ty end points where stream
enter and leave lakes and
the lakes label point
cost_path =
costpath(ends,cost_grid,back
_link,bycel)
```

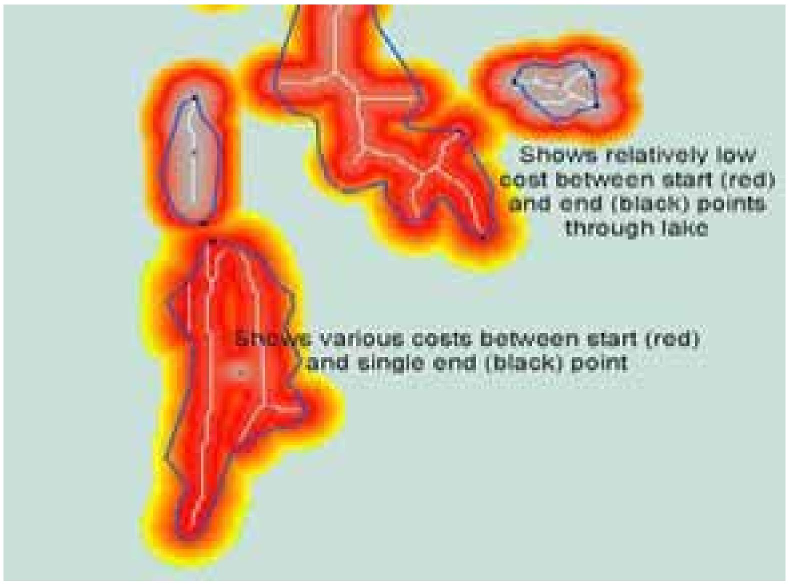

Shows relatively low cost between start (red) and end (black) points through lake
Shows various costs between start (red) and single end (black) point

Figure 16: Cost distance result between start and end points.

The last step in the costpath functions is to convert the spider web of internal lake grid cell paths to vectors. Again we use the gridline functions used for that purpose, Figure 19.

```
&ty Step 7, create the inside lake vectors from the cost_path analysis
lake_path = gridline(cost_path,data,nothin,nofilter,sharp,#,#,#,15)
quit /* Quit out of Grid
```

Now that the arcs through the lakes are present, they must be recombined with the original stream segments water_cl to produce the basic stream network skeleton.

```
&ty Step 8, append
together the inside
lake_paths with the
streams network,
water_cl
copy water_cl stream_seg
dropitem stream_seg.aat
stream_seg.aat mapid id
lake_bnd
build stream_seg line
dropitem stream_seg.aat
stream_seg.aat lake_bnd
append joined_lines line
all
lake_path
```

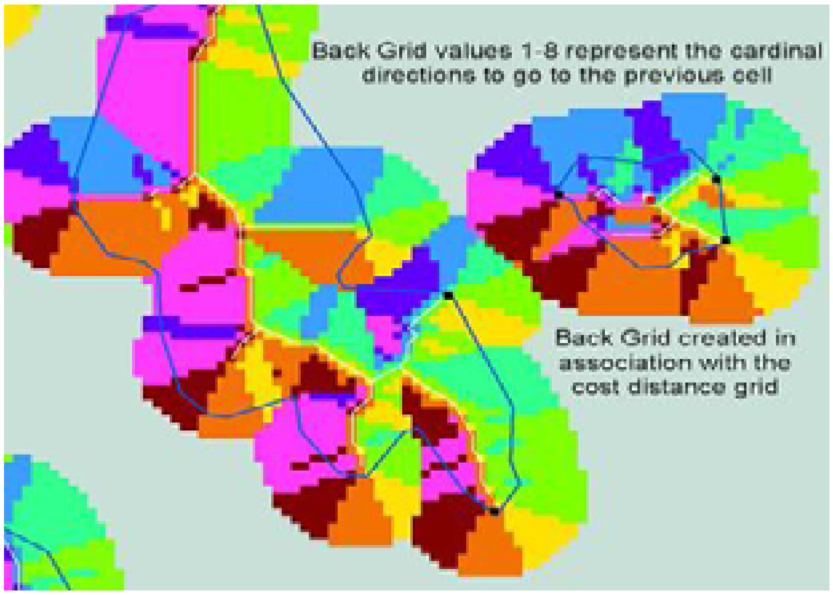

Back Grid values 1-8 represent the cardinal directions to go to the previous cell
Back Grid created in association with the cost distance grid

Figure 17: Back grid shows the cardinal direction to get to the previous cell.

```
stream_seg
end
build joined_lines line
```

The last step in cost_path.aml is to snap the internal lake arcs to the terminal nodes of the incoming and outgoing stream arcs. This is accomplished in a iterative fashion, by snapping terminal nodes together at the lake edge where the streams come in or go out, with ever increasing tolerances, starting at 5m and work up to 25 metres. Experimentation was used to find the best mix of parameters and steps to use. If the grid cell size is altered the values must be altered as well.

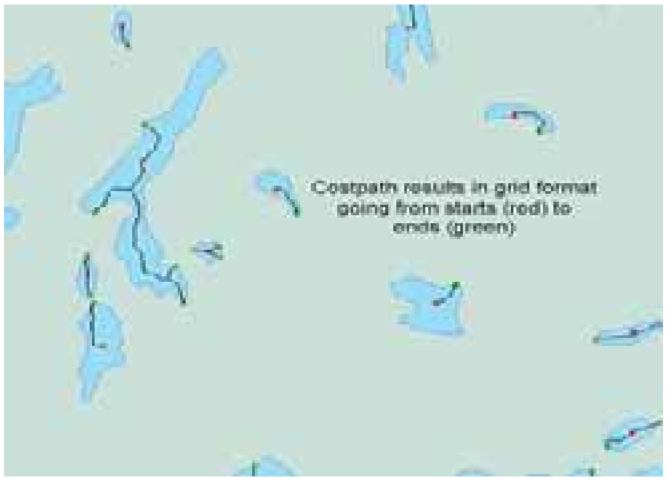


Figure 18: Costpath links between start cells and end cells in raster format.

```
&ty Step 9, cleaning line work to snap
to input and output streams
intersection points matchnode joined_lines matchnode5 5 lake_ends extend
build matchnode5 line
matchnode matchnode5 matchnode10 10
lake_ends extend
build matchnode10 line
matchnode matchnode10 matchnode15 15
lake_ends extend
build matchnode15 line
matchnode matchnode15 matchnode20 20 lake_ends extend
build matchnode20 line
matchnode matchnode20 matchnode25 25 lake_ends extend
build matchnode25 line
```

The final matchnode coverage at 25 metres is renamed to w_arcs_cl, build as line and arcs are unsplit.

```
rename matchnode25 w_arcs_cl
build w_arcs_cl line
arcedit
editcov w_arcs_cl
editf arc
sel all
unsplit
save
quit
build w_arcs_cl line
&return
```

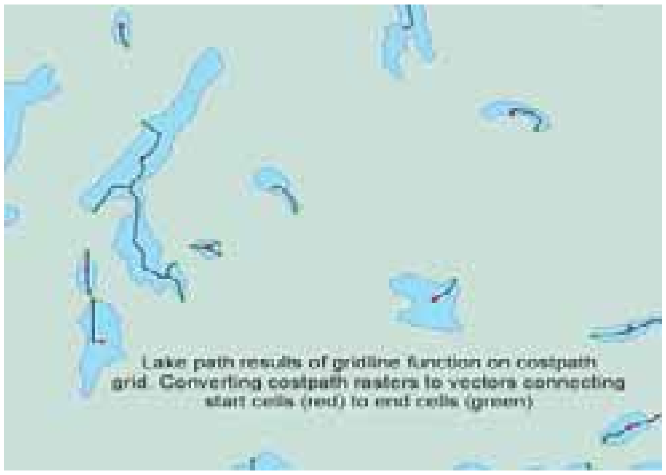

Lake path results of gridline function on costpath
grid. Converting costpath rasters to vectors connecting
start cells (red) to end cells (green)

Figure 19: Gridline result from converting costpath grid values to vectors.

This is where all the automated processing ends and the human interaction of editing begins, The final line data set is w_arcs_cl. See the next section of a continued description.

Cost_path.aml Big Lakes Routine

This is the starting point for the routine buffer called from the top of this aml code.

```
&call buffer
```

As stated its purpose is to add extra arcs to ensure that the large lake polygon label points are within 105m of the cost_grid surface.

```
&routine buffer
```

It starts by looking for a coverage called big lakes to kill it if it exists.

```
&if [exists big_lakes -cover] &then &do
kill big_lakes all
&end
```

ArcEdit is used to create the big_lakes data. Both the outside polygons and the label points are used to create the extra internal arcs.

```
arcedit
editcov sel_lake_dis
editf poly
asel all
reselect area > 2400000 /* metres
put big_lakes
asel all
editf label
asel all
reselect area > 2400000 /* metres
put lglstarts
quit
```

So if a big_lakes data set exists internal buffers are applied to it at different distances, then built as a individual polygon coverages.

```
&if [exists big_lakes -cover] &then &do
```

Extra values are present i.e. commented out below for watersheds that have some large lakes. In Saskatchewan really large lakes meant > 7 kms across by the same long. It was not necessary to have all the buffer values applied since 80% of the watersheds that were processed in Saskatchewan worked well with buffers < 2000m.

```
&do buffname &list 200 400 600 800 1000 1200 1400 1600 1800 2000 2200 2400 2600 2800
3000 5000 7000 10000 15000 20000 30000
&if [exists buff%buffname% -cover] &then &do
kill buff%buffname% all
&end
&end
build big_lakes poly
buffer big_lakes buff200 # # -200 0.001 poly # #
buffer big_lakes buff400 # # -400 0.001 poly # #
buffer big_lakes buff600 # # -600 0.001 poly # #
buffer big_lakes buff800 # # -800 0.001 poly # #
buffer big_lakes buff1000 # # -1000 0.001 poly # #
buffer big_lakes buff1200 # # -1200 0.001 poly # #
buffer big_lakes buff1400 # # -1400 0.001 poly # #
buffer big_lakes buff1600 # # -1600 0.001 poly # #
buffer big_lakes buff1800 # # -1800 0.001 poly # #
buffer big_lakes buff2000 # # -2000 0.001 poly # #
```

The following buffer commands are used when very large lakes exist in a watershed.

```
/* buffer big_lakes buff2200 # # -2200 0.001 poly # #
/* buffer big_lakes buff2400 # # -2400 0.001 poly # #
/* buffer big_lakes buff2600 # # -2600 0.001 poly # #
/* buffer big_lakes buff2800 # # -2800 0.001 poly # #
/* buffer big_lakes buff3000 # # -3000 0.001 poly # #
/* buffer big_lakes buff5000 # # -5000 0.001 poly # #
/* buffer big_lakes buff7000 # # -7000 0.001 poly # #
/* buffer big_lakes buff10000 # # -10000 0.001 poly # #
/* buffer big_lakes buff15000 # # -15000 0.001 poly # #
/* buffer big_lakes buff20000 # # -20000 0.001 poly # #
/* buffer big_lakes buff30000 # # -30000 0.001 poly # #
```

The created internal buffers are built as line datasets.

```
&do buffname &list 200 400 600 800 1000 1200 1400 1600 1800 2000 2200 2400 2600
2800 3000  5000 7000 10000 15000 20000 30000
&if [exists buff%buffname% -cover] &then &do
build buff%buffname% line
&end
&end
```

The internal buffers are appended into a line data set for use later.

```
append buffers line all
buff200
buff400
buff600
buff800
buff1000
buff1200
buff1400
buff1600
buff1800
buff2000
/* buff2200
/* buff2400
/* buff2600
/* buff2800
/* buff3000
/* buff5000
/* buff7000
/* buff10000
/* buff15000
/* buff20000
/* buff30000
end
build buffers line
```

The item value is dropped from water_lclp so that the line append to come will function.

```
dropitem water_lclp.aat water_lclp.aat value
```

As stated, the label points are also used to add extra arcs. This is accomplished by first deleting old buffer coverages if they exist, then buffering large lake label points with progressively larger buffers. All of which are appended together to form a line data set of concentric rings around large lake label points.

```
&do buffnameb &list 100 200 300 400 500 600 700 800 900 1000 1200 1400 1600 1800
2000
&if [exists sbuff%buffnameb% -cover] &then &do
kill sbuff%buffnameb% all
&end
&end
buffer lglstarts sbuff100 # # 100 0.001 point # #
buffer lglstarts sbuff200 # # 200 0.001 point # #
buffer lglstarts sbuff300 # # 300 0.001 point # #
buffer lglstarts sbuff400 # # 400 0.001 point # #
buffer lglstarts sbuff500 # # 500 0.001 point # #
buffer lglstarts sbuff600 # # 600 0.001 point # #
buffer lglstarts sbuff700 # # 700 0.001 point # #
buffer lglstarts sbuff800 # # 800 0.001 point # #
buffer lglstarts sbuff900 # # 900 0.001 point # #
buffer lglstarts sbuff1000 # # 1000 0.001 point # #
buffer lglstarts sbuff1200 # # 1200 0.001 point # #
buffer lglstarts sbuff1400 # # 1400 0.001 point # #
buffer lglstarts sbuff1600 # # 1600 0.001 point # #
buffer lglstarts sbuff1800 # # 1800 0.001 point # #
buffer lglstarts sbuff2000 # # 2000 0.001 point # #
&do buffnameb &list 100 200 300 400 500 600 700 800 900 1000 1200 1400 1600 1800
2000
&if [exists sbuff%buffnameb% -cover] &then &do
build sbuff%buffnameb% line
&end
&end
append lgstartbuffs line all
sbuff100
sbuff200
sbuff300
sbuff400
sbuff500
sbuff600
sbuff700
sbuff800
sbuff900
sbuff1000
sbuff1200
sbuff1400
sbuff1600
sbuff1800
sbuff2000
end
build lgstartbuffs line
```

The appended data are clipped by the first internal lake buffer buff200 to trim off arcs that fall outside the lake boundary. For instance, if a lakes is 500m wide and we buffer its label point 600m than arcs will appear beyond the edge of the lake. In the process we are only concerned about the line in the lake hence the clip by buff200.

```
clip lgstartbuffs buff200 lg_st_buf_cl line 0.001
```

At this point the aml appends together all of the arc data sets. Water_lclp which came from create_thin_lines_(watershed_name).aml and the newly created and append large lake internal buffers plus the large lakes label points which were buffered, appended and clipped.

```
append buff_watline line all
lg_st_buf_cl
water_lclp
buffers
end
```

One issue still remains, the internal lake buffers and label point buffers extend over islands. So we used a buffered version of the island data sets to erase those arcs that traversed islands.

```
buffer islands islands_buff # # 20 0.001 poly # #
build islands_buff poly
erase buff_watline islands_buff buff_watclp line 0.001
```

The final coverage which continues down the cost_path.aml is buff_watclp after it is cleaned and built as line.

```
clean buff_watclp buff_watclp 0.001 0.001 line
build buff_watclp line
For watersheds that do not have large lakes the starting coverage water_lclp is
copied to
buff_watclp so the aml can proceed.
&end
&else
copy water_lclp buff_watclp
&return
```

Background for Cleaning and Editing of the Stream Network

General Information

In any large project a lot of time is spent checking the data to ensure their accuracy, and that quality falls inside the bounds of error expected. For this project, a tolerance of 20m at the connection points between the internal lake skeleton and external stream arcs was acceptable. It is unacceptable for any other arcs to move other than those arcs involved in the connection between streams arcs and in lake arcs. Achieving that level of success with this stream network is an average of 85-90% of the connection points. The last 10-15% requires a person with understanding of GIS editing and hydrology to look at the data and make judgement calls while editing. The following is a list of judgement calls that were made while creating the Saskatchewan Stream Network.

Example, unconnected waterbodies with a stream leading in or out of it.
  Is it a internal drainage?
  Does it hook up some where nearby?
Example, closed loops i.e. a polygon through a chain of lakes or oxbows that for a complete loop.
  Where can an arc be removed in a lake to break the polygon?
Example, enclosed loops because of man made irrigation canals.
  Which irrigation canal can be split or which segment of an irrigation canal can be removed to break the polygon?
Example, meandering or braided streams and or inland deltas all of which can have numerous paths.
  Which stream segments can be broken to develop the best main path through a braided stream network?
  For deltas, which segment should be left to get a main path?

The above consists of 60% of the judgement calls that were made while editing the network. The remaining 40% were just errors in processing. Specifically, sections of in lake lines that could not connect to each other in processing because of the number of islands or the space between islands was smaller than 15 m which made the cost of going between them extremely high hence no path was created.
Most judgment calls were answered by looking at Landsat 7TM data from 1999 - 2001 as a three colour composite image while editing. It was easy to see connections between lakes where arcs just stopped or areas where water was not visible at the time of NTS50 compilation but is visible now.

Process for Stream Network Editing, Dana Jones, Junior GIS Analyst

Abstract

The following document describes the basic step-by-step processes, and related rationale of the methods used to edit the Saskatchewan Stream Network for Saskatchewan Environment. Information Management Branch (IMB) objective was to create a continuous stream network to meet national standard. Example stream network data can be viewed at http://gisweb1/stream_network.

Stream network edits were conducted on both line and polygon feature layers and are outlined in the methodology.

Acknowledgements

I wish to thank Will Stafford for giving me the opportunity to work on this project and for his efforts in establishing the software platforms and setting up the necessary communications protocols to allow the work to be completed.

Introduction

The projected time required for the Watershed Network project was scheduled for approximately 2 years of one person. Saskatchewan Environment (SE) purchased the NTS 1 :50000 base data from the Information Service Cooperation (ISC). SE's Information Management Branch employees objective was to create a continuous stream network to meet the national standard. As of November 2003 all of the province has been networked (see http://gisweb1.serm.gov.sk.ca/mapserver/SDUG/Hydrology%20Project%20Updates.htm).

Data Processing

Startup

Additional coverages were required for editing the stream network data coverage called edit_arcs. Edit_arcs originated from the National Topographic Series 1:50,000 data located on the data server.

Table 2. Coverages used for editing and their description

Coverage Name	Description
edit_arcs	stream network line coverage
sel lake dis	waterbody outlines
water_cl buf	2 meter watershed buffer
islands	island outlines
neats 50	NTS 50 neatlines
Landsat7	Landsat 7 TM data deisplayed as a 3 colour composite

The watershed project was edited using ArcInfoTM 8.0. The AML (arc_check_new.aml) was used to create new coverages (dangle_arcs, arc_gap_fix, shordngls, wateclbuf), which allowed the user to identify errors. (see arc_check_new.aml). The bash command shell was used to identify and locate the current workspace, and the amls were run inside the arc shell.

```
Example
Arc: &r arc_check_new.aml
```

Arc_check_new.aml is an aml program that creates new coverages which are used by the editor to help him/her with finding errors and then editing them.

The watershed was edited by executing the following commands from the Arc: prompt:

```
Arc: ae /* prompts the ArcEdit command shell
Arcedit: display 9999 /* opens the display window
Arcedit: mapextent editarcs /* sets map extent of the display window
Arcedit: editcoverage editarcs /* sets the edit coverage
Arcedit: editfeature arcs * / sets the feature type
Arcedit: drawenvironment arcs /* sets the draw environment features
Arcedit: draw /* draws the coverages and their respective features
Arcedit: intersectarcs all /* intersect all new added arcs to existing arcs
Arcedit: nodesnap closest 20 /* node snapping will search the entire area inside the
NODESNAP distance from the node under consideration and snap to the closest node
found.
Arcedit: drawenvironment node dangle /* displays dangles
Arcedit: nodecolor dangle 2 /* displays dangles as red
Arcedit: drawenvironment node errors /* displays errors
Arcedit: nodecolor pseudo 3 /* displays errors (pseudo) as green
Arcedit: weedtolerance /* specifies minimum distance between adjacent vertices on an
added arc.
Background coverages were used for reference while editing the watershed coverage
(editarcs).
Arcedit: backenvironment arc /* sets background feature environment (arc)
Arcedit: backcoverage sel_lake_dis 4 /* sets background lakes in blue
Arcedit: backcoverage watecl_buf 7 /* sets background 2 meter buffer yellow
Arcedit: backcoverage neats_50 /* sets background NTS 50 neatlines white
Arcedit: backcoverage islands 8 /* sets background islands orange
Arcedit: build line /* updates node, line and/or polygon topology
```

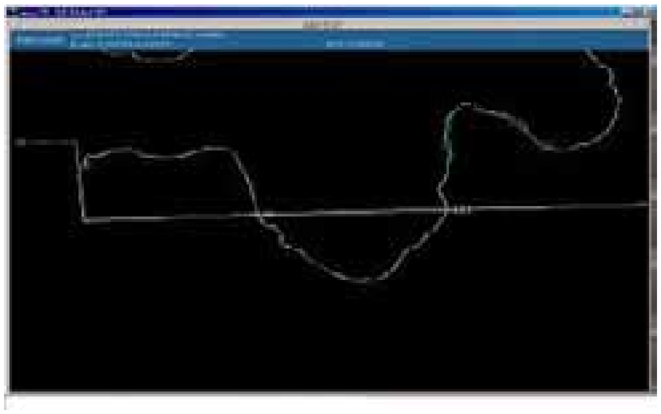
```
Arcedit: clean 0.001 0.001 /* cleans the edit coverage with a fuzzy tolerance of 1
millimeter only if it is required or prompt. Clean can be undone with the "oops"
command

Arcedit: draw
```

Note: Watershed_startup.aml was used to efficiently open each project without manually typing each commands listed above.

Removing Polygons

Once the data had been drawn on screen, series of steps were completed in order to edit the watershed data. The watersheds are line coverages, therefore polygons labeled after the build command must be deleted. Displaying the polygons label point and id numbers minimized the difficulty of identifying polygons.

```
Arcedit: drawenvironment
labels ids /*set draw
environment labels
Arcedit: draw /*draws the
data set w/ polygon label
points and id numbers
```



When deleting a polygon, proper judgment was used to ensure the correct arcs were selected and removed. The main focus while editing a polygon from a stream network was to remove as little as possible from the coverage (edit_arcs), while maintaining the contiguity of the stream network. Arcs were selected and deleted to remove a polygon provided that the network flow was not permanently cut off to a particular location. Arcs in a stream network were either man made or natural. When deciding which channel to remove, man-made channels were chosen over natural flowing, Figure 20.

Editing Man-Made Channel Polygons

If man-made channels formed a polygon, arcs were split to form gaps approximately 50 meters apart. To split up the man-made channel, add two short arcs that perpendicularly intersect the arc to be split, Figure 21.

```
Arcedit: add /* adds arcs to edit
coverage.
```

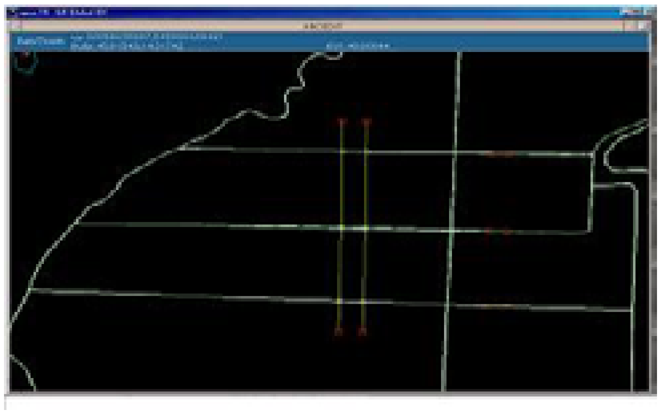


Figure 21: Splitting man-made channels.

After the two arcs were added, they were selected including the 50 meter arcs in between the added lines. Once all the arcs were selected, the delete command was used to create a gap in between the man-made channels, Figure 22.

```
Arcedit: sel /* selects editfeature of the edit coverage
```

```
Arcedit: delete /* deletes a selected feature
```

After deleting the arcs, a 50 meter gaps was left in the manmade channel, which opened up the polygon, Figure 22.

Editing Natural Channel Polygons

Selecting and deleting the shortest arc to break open the polygon removed natural stream flow polygons, Figure 23.

```
Arcedit: sel
Arcedit: delete
```

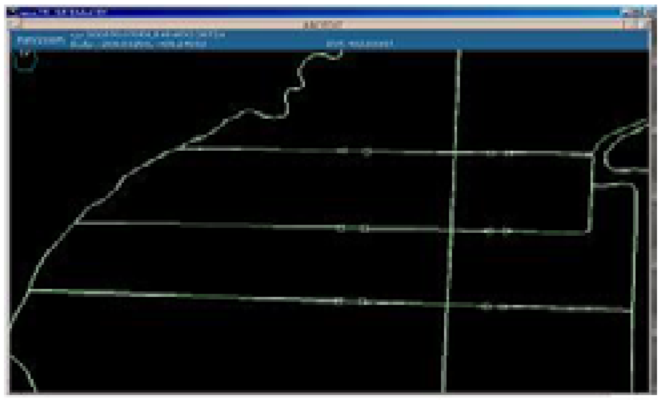


Figure 22: 50-meter gaps in man-made channels.

When deciding which polygon arc to remove, the shortest arcs were selected and deleted, providing that the network flow was not permanently cut off to a particular location. Zooming out from the polygon allowed for a clearer perspective of the stream network and its flow characteristics. After the polygons arc was deleted, the flow of the network was followed through the edited polygon. If the flow was not impeded as a consequence of the edit, then the edit was considered complete.

Editing lines

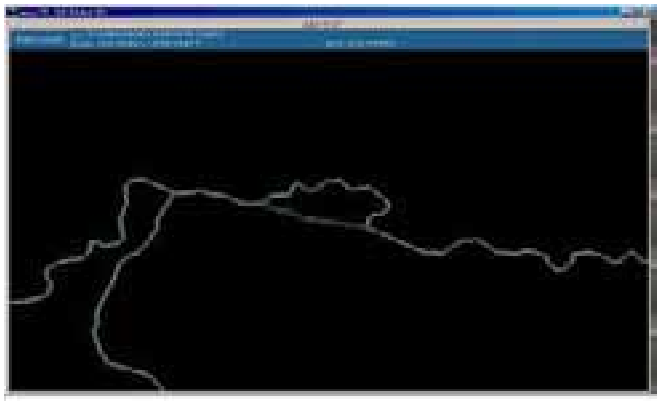


Figure 23: Short and long arcs.

ArcView was used as a tool to find potential errors that could not be visually discovered in ArcInfo. Open up an Arc View session and add the same coverages that were added to your ArcInfo session. In ArcView, dangle_arcs was converted to a shape file by choosing the menu: Theme Convert to Shapefile, Figure 24. Start editing the new coverage by choosing: Theme Start Editing. These errors are much easier to display visually while the coverage is in edit mode, Figure 25. Once in editing mode, the attributes of the shapefile were easily found on screen using the feature selector button. The pointer arrow was used by clicking and dragging a box across the full length of the graphics display window. Selecting the attributes of the selected shapefile displayed the attributes, which allowed for easier error detection, Figure 25. Once the potential errors were found in ArcView, switching over to the ArcInfo session and zooming into the same location identified the potential errors and a decision was made. Correct the error if needed, note: some errors did not need to be fixed.

Correcting the Error

The watershed line coverage was edited using both Arc View and ArcInfo. Arc View was used to visually locate the errors, refer to Editing Lines, while ArcEdit was used to correct the errors. ArcEdit was chosen as the editing program because of its stability, speed, and superior editing functions, while ArcView is know for its graphic displaying qualities. Arcs were edited using a combination of commands such as select, split, delete, and add.

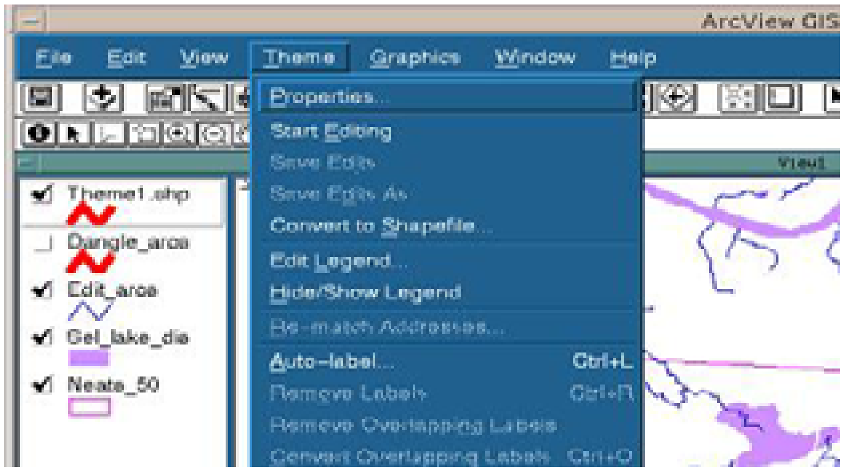


Figure 24: Shapefile conversion and editing mode.

The five common ArcEdit commands used while editing the watershed arcs were:

```
Arcedit: sel
Arcedit: split /* Splits
the arc by left clicking
with the mouse
Arcedit: sel
Arcedit: delete
Arcedit: add
```

Most arcs were selected, deleted, then added, but longer arcs needed to be selected and split (creating pseudo nodes) before deleting, Figure 26.

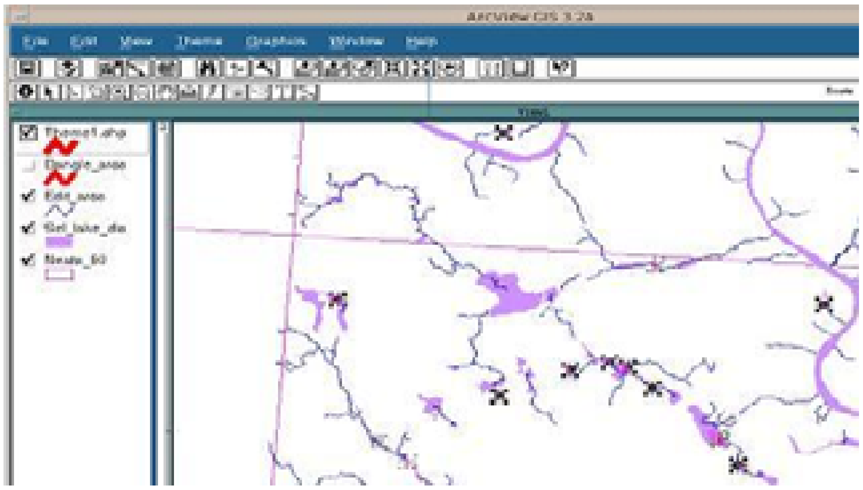


Figure 25: Highlighted shapefile errors.

Restoring previously deleted arcs back into the flow network

Arc View was used to determine if arcs in the ms_arcs_put coverage were missing from the edit_arcs coverage. In ArcView, arcs were compared between the two coverages and ms_arcs_puts arcs were selected if missing. Once the arcs were selected in ms_arcs_put, the query function was used to calculate tables field put = 1. The table must be in the edit mode and read/write privileges must be granted in order to calculate the table. After the edits were saved in ArcView, ArcInfo was used to put the missing arcs into the editarc coverage.

```
Arcedit: editcoverage ms_arcs_put /* the edit coverage will be the newly created
shapefile
from ArcView. This was the coverage with the selected arcs.
```

```
Arcedit: editfeature arc
Arcedit: aselect all /* all the ms_arcs_put arcs are selected
Arcedit: reselect put = 1 /* The arcs with a put value of 1 are selected
Arcedit: put = 1 /*Creates a new value of 1 for the item put
Arcedit: put editarcs /* Appends the selected arcs to the desired coverage
Arcedit: q
```

Locating and editing dangles and restoring gaps between arcs

The arc_check_new.aml created the coverages short dangles and arc_gap_fix, which were used to locate possible errors. The coverage short dangles displayed dangles, which are less than 25 meters. The coverage arc_gap_fix displayed arc gaps, which are less than 30 meters. The short dangles and arc_gap_fix coverages should be checked using similar methods described in Editing Lines.

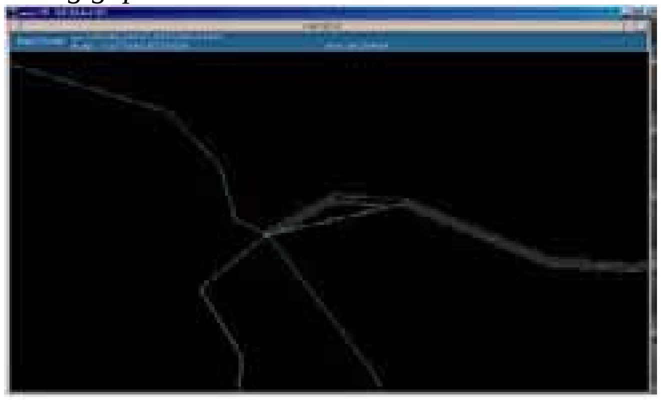


Figure 26: Arc pseudo nodes.

Time Requirements

For future reference, the time required to complete a watershed, with an average of 13000 arcs and 400 polygons, took approximately 1 week to complete. This assessment is made based on the assumption that one GIS Analysts is working on the network full time (8 hour day, 5 days week).

Additional Problems

Beginning a new watershed required the ownership of the directory to be changed from the creator to the person editing the data. If ownership was not changed, the coverages could not be edited or added. As a consequence of using large data sets, drive space posed a problem. In order to free up drive space, temporary files not needed were deleted using either the home folder in Opus or Sockets File Transfer Protocol (WS_FTP) program to access the files and remove them from the watershed's workspace.
In order to run the .amls, make sure the workspaces have all the coverages required, and coverage names are synonymous with the names specified in the .aml. If coverages become corrupted, make a copy of the coverage in the same directory and attempt the task again. If workspaces become corrupted, copy the coverages to a new workspace.

Glossary

Build - creates or updates a feature attribute table for a coverage, generates or updates polygon topology and the polygon attribute table without correcting coordinate errors. BUILD also adds label points if no label exists.

Clean - generates a coverage with correct polygon or arc-node topology. To do this, CLEAN edits and corrects geometric coordinate errors, assembles arcs into polygons and creates feature attribute information for each polygon or arc.

Intersectarcs all: specifies that arc intersections will be calculated and nodes added for all operations that edit arc coordinates. This includes ADD, COPY, ROTATE, and others.

Nodesnap closest: Node snapping ensures that edited arcs connect to nodes of existing arcs in the coverage. This is important for closing polygons and correcting arc overshoots or undershoots.

Node snapping occurs when an arc is added or the coordinates of a node change and the node is within the node snap distance of another node. For example, when an arc is added or moved, the nodes may snap to an existing node in the edit coverage. When the search method is set to FIRST, the node will snap to the first edit coverage node found within the search tolerance. This is slightly faster in execution, but if there is more than one node within the NODESNAP distance, the first node may not be the closest, and nodes may snap to the wrong node. NODESNAP CLOSEST avoids this problem and is not noticeably slower.

Put - copies selected features and their attributes into a coverage, or selected INFO records into an INFO file. PUT places copies of the selected features in the edit coverage into <cover>.

Weedtolerance - specifies the minimum distance between vertices for added arcs.

JUMP Skeltonizer, another Lake/River Skeltonizer Tool

Upon completing Version 2 of this project, I was given access to experimental code that used a Voronoi skeleton to create internal lake lines (skeleton). It is a plug into a JAVA based GIS called the Jump Project, JUMP Unified Mapping Platform (JUMP) developed by Vivid Solutions, Inc. and Refractions Research Inc. (http://www.jump-project.org/). It is free software for all to use and can do very large data sets. Here is another option to create internal lake skeletons.
This specific plugin and others can be downloaded at:

http://www.jump-project.org/portal.php?PID=PL

The product works as claimed and can do some unique spatial analysis. Given that this tool reads writes ESRI shapefiles if I had to do this all over again this is how I would do it.

Contact Information

Will Stafford, GIS Consultant
Po Box 472
Prince Albert Saskatchewan
Stafford.will@sasktel.net

Dana Jones, GIS Analyst
Saskatchewan Environment, Fire Management
Po Box 3003
Hwy #2 North
Prince Albert Saskatchewan
S6V 6G1
Dana.jones@gov.sk.ca

Appendix 1: ArcInfo Processing amls

Appendix 1a: create_thin_lines_(watershed_name).aml

First aml to run is create_thin_lines_(watershed_name).aml, the example listed is:
create_thin_lines_kasba_lake.aml

```
/*******************************************************************
/*    This aml was developed and compiled by Will Stafford, Information Management
/*    Branch
/*    Saskatchewan Environment.
/*    It is one of 3 amls used to develop a stream network for the Province of
Saskatchewan,
/*    based on the National Topographic Series 1:50,000 Data from Natural Resources
/*    Canada.
/*    This aml or parts there of can be freely distributed, however recognition of the
author
/*     in code or product would be appreciated.
/*     The resultant loss of data or hassles associated with running this aml is the
/*     responsibility of the user and not the author or any individual associated the
/*     Saskatchewan Government.
/*    If you require assistance in using this aml or desire knowledge or understanding
of
/*     the thought processes behind this aml please contact;
/*    Will Stafford
/*    Information Management Branch, Saskatchewan Environment
/*    Prince Albert, Saskatchewan
/*    306 953 2219
/*    Other Contact
/*    Ken Yurach
/*    Information Management Branch, Saskatchewan Environment
/*    Prince Albert, Saskatchewan
/*    306 953 2465
/*    Date December 2, 2003
/*******************************************************************
/* Create a watch file of processes run
&watch create_thin_lines.txt
&ty       Killing of temp files if exists
 &call kill
/* Set precision to double double for workspace
precision double double
/*************************************
/* Buffer stream network for selecting which lakes are part of stream network
/* Set to 10 metres
buffer water_cl strm_buff # # 10 0.001 line round ful
/* clip lake with stream buffer to select out lakes touching streams
clip water_ba strm_buff lake_clp poly 0.001
build lake_clp poly
copy water_ba lake_2
build lake_2 poly
/* Calc lake-id for join
additem lake_2.pat lake_2.pat lake-id 4 5 b
additem lake_clp.pat lake_clp.pat lake-id 4 5 b
tables
```

```
sel lake_2.pat
calc lake-id = lake_2-id
sel lake_clp.pat
calc lake-id = lake_clp-id
quit
joinitem lake_2.pat lake_clp.pat lake_2.pat lake-id # linear
/* Save out lakes that touch streams
arcedit
editcov lake_2
editf poly
sel all
reselect lake_clp-id > 0
put sel_lakes
quit
build sel_lakes poly
copy sel_lakes tmp_lakes
build tmp_lakes poly
/* Select out islands to create a separate island coverage
arcedit
editcov water_ba
editf poly
asel all
reselect atc = 0
put islands
quit
build islands poly
/* Dissolve away islands inside lakes
tables
sel tmp_lakes.pat
reselect area > 0
calc atc = 1450
quit
dissolve tmp_lakes sel_lake_dis atc poly
/* Make sure all polygons have a label point
createlabels sel_lake_dis
build sel_lake_dis poly
build sel_lake_dis line
/* Create holes in lake grid to avoid gridline errors
arcedit
editcov sel_lake_dis
editf poly
asel all
reselect area > 50000
put sel_lake_net
quit
build sel_lake_net poly
buffer sel_lake_net net_lake_ins # # -100 0.001 poly # #
build net_lake_ins poly
/* USER INTERVENTION AREA, START
/*************************************************************************
/*************************************************************************
/* USER must adjust folowing values to create the "lake holes" data set
/* Develop routine to create fishnet from coverage bnd
/* generate net /* base parameters
/* fishnet coordinate (x,y)
/* y-axis origin (x,y)
/* cel size (width,height)
/* number of rows,columns
generate net /* Set up for Kasba Lake Watershed
fishnet
573000,6565900
573000,6656200
200,200
451,500
quit
build net point
/***********************************************************************
/***********************************************************************
/ *END OF USER INTERVENTION AREA
clip net net_lake_ins net_clp point #
buffer net_clp net_clp_buf # # 30 # point # #
additem net_clp_buf.pat net_clp_buf.pat cel_v 5 5 i
tables
sel net_clp_buf.pat
calc cel_v = 1
quit
/* Enter grid to convert lines and polygons to grids
grid
lake_grid = polygrid(sel_lake_dis,sel_lake_dis#,#,#,10)
stream_g = linegrid(water_cl,water_cl#,#,#,10,nodata)
island_g = polygrid(islands, islands#,#,#,10)
net_grd = polygrid(net_clp_buf,cel_v,#,#,10)
/* Burn in islands as no data values to lake_grid
lake_isl_grd = con(isnul(island_g), lake_grid, setnul(island_g))
/* join lake_grid and stream_g together
lake_strm_grd = con(isnul(stream_g),lake_isl_grd,stream_g)

/* join lake stream grid with net_grid to punch holes on large lakes
stream_line = con(isnul(net_grd),lake_strm_grd,setnul(net_grd))
/* create water courses through lakes and streams as continuous set of lines
water_l = gridline(stream_line,data,thin,nofilter,round,value,#,#,#)
/* POSSIBLE USER INTERVENTION AREA IF GRIDLINE ABOVE FAILS, START
/***********************************************************************
/***********************************************************************
/* Develop routine here to determine coverage splitting
/* magic number is 16000 by 17000 pixels
/* If gridline dies split stream_line into east west halves via arcview
/* The first step was to create a square boundary around the whole watershed.
/* I created the boundary in ArcView 3.2a because of its ease of use.
/* The resulting polygon shapefile was converted to a coverage.
/* You could just as easily create the square boundary in ArcEdit from the get go,
/* but ArcView was just so easy Figure 10.
/* So in ArcView add the watershed boundary.
/* Create a new polygon theme, View, New Theme, Polygon, caled square_bnd, Okay.
/* Give it a name and put it in the stream network workspace.
/* Select the box symbol and create a square polygon.
/* Its okay to go several killometres larger than the watershed. Theme Stop Editing,
Save
Edit Yes.
/* From this point on all of the editing done to create those clipping coverages
/* is done in ArcInfo's ArcEdit module after you convert the shape file into a polygon
cover-age.
shapearc square_bnd sq_bnd
build sq_bnd poly
arcedit
editcoverage sq_bnd
display 9999
drawenvironment arcs
draw
editfeature arc
intersecarcs all
add /* add a vertical line down the middle of sq_bnd to split it into two polygons with
large
/* dangles for ease of editing
select many /* select and delete the dangles from the added arc
delete
build /* build the data as a polygon
editfeature  poly
select one /* select the east polygon i.e. one
put one
sel one /* select the west polygon i.e. two
put two
quit
n /* don't save changes to sq_bnd keep it as the ordinal one polygon boundary
/* Now we move into grid to clip stream_line into two data sets and try gridline
/* to see if it will complete with only the one cut.

grid /* start grid
gridclip stream_line strm_one cover one
gridclip stream_line strm_two cover two
one_l = gridline(strm_one,data,thin,nofilter,round,value,#,#,#)
two_l = gridline(strm_two,data,thin,nofilter,round,value,#,#,#)
quit /* quit grid
/* If all is wel i.e. gridline is able to create the two vector coverages one_l and two_l
/* than append the two data sets together to create water_l and move on to the next
section.
append water_l line all
one_l
two_l
end
/* The following code details two more splits which create 4 data sets that might be
needed
/* for a successful conversion of the grid data to vector data. all of which need to be
/* append together to form the final vector coverage. In Saskatchewan the Churchwest
watershed
/* northwest section required six parts for it to work.
/* This looks after the one polygon coverage.
arcedit
editcoverage one
display 9999
drawenvironment arcs
draw
editfeature arc
intersecarcs all
add /* add a vertical line down the middle of one to split it into two polygons with
/* large dangles for easy of editing
select many /* select and delete the dangles from the added arc
delete
build /* build the data as a polygon
editfeature  poly
select one /* select the east polygon i.e. one
put one1
sel one /* select the west polygon i.e. two
put one2
quit
build one1 poly;build one2 poly
/* This looks after the two polygon coverage.
arcedit
editcoverage two
display 9999
drawenvironment arcs
draw
editfeature arc
intersecarcs all
add /* add a vertical line down the middle of one to split it into two polygons with
/* large dangles for ease of editing
select many /* select and delete the dangles from the added arc
delete
build /* build the data as a polygon
editfeature  poly
select one /* select the east polygon i.e. one
put two1
sel one /* select the west polygon i.e. two
put two2
quit
build two1 poly;build two2 poly
/* Now enter grid and start work there for one1, one2
grid /* start grid
gridclip stream_line strm_one1 cover one1
gridclip stream_line strm_one2 cover one2
one1_l = gridline(strm_one1,data,thin,nofilter,round,value,#,#,#)
one2_l = gridline(strm_one2,data,thin,nofilter,round,value,#,#,#)
quit /* quit grid
/* Now enter grid and start work there for two1, two2
grid /* start grid
gridclip stream_line strm_two1 cover two1
gridclip stream_line strm_two2 cover two2
two1_l = gridline(strm_two1,data,thin,nofilter,round,value,#,#,#)
```

```
two2_l = gridline(strm_two2,data,thin,nofilter,round,value,#,#,#)
quit /* quit grid
/* Now append together the vector results to create water_l
append water_l line all
one1_l
one2_l
two1_l
two2_l
end
/* Lastly clean and build the water_l data set. The water_l was cleaned
/* with a movement of 12m which is 2 metres greater than the source raster data set
cell size
build water_l line
clean water_l water_l_cl 10 12
build water_l_cl line
/* The data set water_l_cl is final result to create_thin_lines_(watershed_name).aml.
/* It is used in the cost_path.aml.
quit /* Leave Grid
/**********************************************************************
/**********************************************************************
/* END OF USER INTERVENTION AREA
/* build topology for water_lines lines
build water_l line
/* Clean water_l with a movement of 12m
clean water_l water_l_cl 10 12
build water_l_cl line
/* clean up files not needed any more
/* Clean up routine if you want to immediately kill all temp data sets.
/*  &call kill2
&watch &off
&return
/* Used to clean up all datasets so you can start anew.
&routine kill
&ty         Killing temp grids at start of aml
/* Kill Grids
&if [exists lake_strm_grd -grid] &then &do
kill lake_strm_grd all
&end
&if [exists net_grd -grid] &then &do
kill net_grd all
&end
&if [exists stream_g -grid] &then &do
kill stream_g all
&end
&if [exists lake_grid -grid] &then &do
kill lake_grid all
&end
&if [exists lake_isl_grd -grid] &then &do
kill lake_isl_grd all
&end
&if [exists island_g -grid] &then &do
kill island_g all
```

```
&end
&if [exists stream_line -grid] &then &do
kill stream_line all
&end
/* Kill Covers
&ty        Killing temp covers at start of aml
&if [exists net_lake_ins -cover] &then &do
kill net_lake_ins all
&end
&if [exists net -cover] &then &do
kill net all
&end
&if [exists sel_lake_net -cover] &then &do
kill sel_lake_net all
&end
&if [exists net_clp -cover] &then &do
kill net_clp all
&end
&if [exists net_clp_buf -cover] &then &do
kill net_clp_buf all
&end
&if [exists water_l_cl -cover] &then &do
kill water_l_cl all
&end
&if [exists sel_lakes -cover] &then &do
kill sel_lakes all
&end
&if [exists strm_buff -cover] &then &do
kill strm_buff all
&end
&if [exists lake_clp -cover] &then &do
kill lake_clp all
&end
&if [exists lake_2 -cover] &then &do
kill lake_2 all
&end
&if [exists lake_poly -cover] &then &do
kill lake_poly all
&end
&if [exists tmp_lakes -cover] &then &do
kill tmp_lakes all
&end
&if [exists sel_lake_dis -cover] &then &do
kill sel_lake_dis all
&end
&if [exists sel_lake -cover] &then &do
kill sel_lake all
&end
&if [exists water_l -cover] &then &do
kill water_l all
&end
&if [exists islands -cover] &then &do
```

```
kill islands all
&end
&return
/* Used to clean up all temporary coverages
&routine kill2
&ty         Killing temp grids at end of aml
/* Kill Grids
&if [exists lake_strm_grd -grid] &then &do
kill lake_strm_grd all
&end
&if [exists net_grd -grid] &then &do
kill net_grd all
&end
&if [exists stream_line -grid] &then &do
kill stream_line all
&end
&if [exists stream_g -grid] &then &do
kill stream_g all
&end
&if [exists lake_grid -grid] &then &do
kill lake_grid all
&end
&if [exists lake_isl_grd -grid] &then &do
kill lake_isl_grd all
&end
&if [exists island_g -grid] &then &do
kill island_g all
&end
/* Kill Covers
&ty    Killing temp covers at end of aml
&if [exists net_lake_ins -cover] &then &do
kill net_lake_ins all
&end
&if [exists net -cover] &then &do
kill net all
&end
&if [exists sel_lake_net -cover] &then &do
kill sel_lake_net all
&end
&if [exists net_clp -cover] &then &do
kill net_clp all
&end
&if [exists net_clp_buf -cover] &then &do
kill net_clp_buf all
&end
&if [exists sel_lakes -cover] &then &do
kill sel_lakes all
&end
&if [exists strm_buff -cover] &then &do
kill strm_buff all
&end
&if [exists lake_clp -cover] &then &do
```

```
kill lake_clp all
&end
&if [exists lake_2 -cover] &then &do
kill lake_2 all
&end
&if [exists lake_poly -cover] &then &do
kill lake_poly all
&end
&if [exists tmp_lakes -cover] &then &do
kill tmp_lakes all
&end
&if [exists sel_lake -cover] &then &do
kill sel_lake all
&end
&if [exists islands -cover] &then &do
kill islands all
&end
&return
/*
```

Appendix 1b: start_end_points.aml

Second aml to run is the start_end_points.aml Its purpose is to create start and end points for the
internal lake lines to join to during the costpath analysis aml.

```
/*********************************************************************
/*    This aml was developed and compiled by Will Stafford, Information Management
/*    Branch
/*    Saskatchewan Environment.
/*    It is two of 3 amls used to develop a stream network for the Province of
Saskatchewan,
/*    based on the National Topographic Series 1:50,000 Data from Natural Resources
/*    Canada.
/*    This aml or parts there of can be freely distributed, however recognition of the
author
/*     in code or product would be appreciated.
/*     The resultant loss of data or hassles associated with running this aml is the
/*     responsibility of the user and not the author or any individual associated the
/*     Saskatchewan Government.
/*    If you require assistance in using this aml or desire knowledge or understanding
of
/*     the thought processes behind this aml please contact;
/*    Will Stafford
/*    Information Management Branch, Saskatchewan Environment
/*    Prince Albert, Saskatchewan
/*    306 953 2219
/*    Other Contact
/*    Ken Yurach
/*    Information Management Branch, Saskatchewan Environment
/*    Prince Albert, Saskatchewan
/*    306 953 2465
/*    Date December 2, 2003
/*********************************************************************
```

```
/* Creates a set of points that are the entrances to lakes
/* that are used as tie  points for the streams coverage and
/* the lines that go through the lakes.
/* Create a watch file of the process run
&watch start_end_points.txt
&ty   Killing of temp files if exists
/* Kill of temp coverages
 &call kill
/* Create a backup of water_cl
&if [exists water_cl_org -cover] &then
   &do
   kill water_cl_org all
   &end
copy water_cl water_cl_org
/* Clear extra items in water_cl if the exist
/* USER INTERVENTION AREA, REMOVE ITEMS THAT DON'T NEED TO BE
THERE
/************************************************************************
/************************************************************************
dropitem water_cl.aat water_cl.aat mapid water_cl_i elevation type accuracy ate atz
atg atc
code_gener id entityname tmp1_id order_id order_
&if [iteminfo sel_lake_dis -line lake_bnd -exists] &then
   &do
   dropitem sel_lake_dis.aat sel_lake_dis.aat lake_bnd
   &end
&if [iteminfo water_cl -line lake_bnd -exists] &then
   &do
   dropitem water_cl.aat water_cl.aat lake_bnd
   &end
&if [iteminfo water_cl -line value -exists] &then
   &do
   dropitem water_cl.aat water_cl.aat value
   &end
/************************************************************************
/************************************************************************
/* END OF USER INTERVENTION AREA
/* Add a relate item between lake edge arcs and stream arcs so each can be
/* identified later
additem sel_lake_dis.aat sel_lake_dis.aat lake_bnd 4 5 n
additem water_cl.aat water_cl.aat lake_bnd 4 5 n
tables
sel sel_lake_dis.aat
asel all
calc lake_bnd = 1000
sel water_cl.aat
asel all
calc lake_bnd = 100
quit
/* Append, clean and build  lake and stream arc data sets
append water_arcs line all
sel_lake_dis
```

```
water_cl
end
clean water_arcs water_arcs 0.001 0.001
build water_arcs line
/* Unsplit lake arcs
arcedit
editcov water_arcs
editf arc
sel all
unsplit
save
sel all
resel lake_bnd = 1000
put lake_bnd
quit
renode lake_bnd
build lake_bnd line
/* Output location of stream entrances
nodepoint lake_bnd lake_ends
build lake_ends point
additem lake_ends.pat lake_ends.pat lake_bnd 4 5 n
/* lake coverage is the node points where a stream and lake intersect
/* Now get labels from dissolved lakes and create a point coverage
/* identifying lake centre points where arcs will start from
/* in costpath analsis
arcedit
editcov sel_lake_dis
editf label
sel all
put lake_labels
quit
build lake_labels point
dropitem lake_labels.pat lake_labels.pat atc
additem lake_labels.pat lake_labels.pat lake_bnd  4 5 n
/* Develop point data set with all nodes that have
/* elevation values
tables
sel lake_labels.pat
calc lake_bnd = 1000
sel lake_ends.pat
calc lake_bnd = 100
quit
append node_elv point all
lake_labels
lake_ends
end
build node_elv point
&watch &off
/*  Can turn on kill2 if you want to immediately kill off temp coverages
/* call kill2
&return
&routine kill
```

```
&ty	Killing temp covers at start of aml
/* Kill covers
&if [exists water_arcs -cover] &then &do
kill water_arcs all
&end
&if [exists lake_labels -cover] &then &do
kill lake_labels all
&end
&if [exists lake_bnd -cover] &then &do
kill lake_bnd all
&end
&if [exists lake_ends -cover] &then &do
kill lake_ends all
&end
&if [exists node_elv -cover] &then &do
kill node_elv all
&end
&return
&routine kill2
&ty	Killing temp covers at end of aml
/* Kill covers
&if [exists water_arcs -cover] &then &do
kill water_arcs all
&end
&if [exists lake_bnd -cover] &then &do
kill lake_bnd all
&end
&if [exists node_elv -cover] &then &do
kill node_elv all
&end
&return
```

Appendix 1c: cost_path.aml

The third aml to run is the cost_path.aml. Its purpose is to create the interlake connections between the stream entrance locations and the lake label locations.

```
/*******************************************************************
/*     This aml was developed and compiled by Will Stafford, Information Management
/*     Branch
/*     Saskatchewan Environment.
/*     It is three of 3 amls used to develop a stream network for the Province of
Saskatchewan,
/*     based on the National Topographic Series 1:50,000 Data from Natural Resources
/*     Canada.
/*     This aml or parts there of can be freely distributed, however recognition of the
author
/*      in code or product would be appreciated.
/*      The resultant loss of data or hassles associated with running this aml is the
/*      responsibility of the user and not the author or any individual associated the
/*      Saskatchewan Government.
/*     If you require assistance in using this aml or desire knowledge or understanding in
/*      the thought processes behind this aml please contact;
/*     Will Stafford
/*     Information Management Branch, Saskatchewan Environment
/*     Prince Albert, Saskatchewan
/*     306 953 2219
/*     Other Contact
/*     Ken Yurach
/*     Information Management Branch, Saskatchewan Environment
/*     Prince Albert, Saskatchewan
/*     306 953 2465
/*     Date December 2, 2003
/*******************************************************************
&watch cost_path.txt
&ty         Killing of temp files if exists
 &call kill
&ty Creating a clipped version of the 15m cleaned water_l_cl that go through a lake
clip water_l_cl sel_lake_dis water_lclp line 0.001
build water_lclp line
 &call buffer
&ty Creating buffers for large lake and intersecting those with the thinnied grid lines
/* of buff_watclp and erase out islands
&ty Create a mask from the selected lakes data set that is 20m larger
buffer sel_lake_dis sel_lake_buf # # 20 0.001 poly round ful
polygrid sel_lake_buf mask # # #
10
y
&ty Start grid for analysis and creation of centre lines
grid
&ty Step 1, need starting point for costpath analysis which
/* is the lake label point for each of the
/* lakes that have streams coming from them i.e. labels
/* from sel_lake_dis and must be integer grid
starts = pointgrid(lake_labels,lake_labels-id,#,#,10,nodata)
start_cels = int(starts)
```

```
&ty Step 2, need end points for costpath analysis
/* which are the locations where streams
/* intersect the lake boundary from selected lakes i.e. sel_lake_dis
ends = pointgrid(lake_ends,lake_ends-id,#,#,10,nodata)
&ty Step 3, get a mask set up to constrain the analysis
/* area to the selected lakes only
quit
grid
setmask mask
&ty Step 4, create line distance grid which becomes the
/* line_distance grid from edge of shore to a line
cost_grid = linedist(buff_watclp,buff_watclp,10,dist,105)
/* USER INTERVENTION AREA, MAY HAVE TO SPLIT buff_watclp
/************************************************************************
/************************************************************************
/* When file size exceeds 2.14 gigs split buff_watclp into two parts
/* arcedit
/* editcov sq_bnd /* May already exists if intervention was required in create_thin_line
/* editf arc
/* intersectarcs all
/* drawenv arcs node
/* add
/* sel many
/* delete
/* build
/* editf poly
/* sel one
/* put one /* May already exist if intervention was required in create_thin_line
/* sel one
/* put two /* May already exist if intervention was required in create_thin_line
/* quit
/* build one poly
/* build two poly
/* clip buff_watclp one buff_watclp1 line 0.001
/* clip buff_watclp two buff_watclp2 line 0.001
/* cost_grid1 = linedist(buff_watclp1,buff_watclp1,10,dist,105)
/* cost_grid2 = linedist(buff_watclp2,buff_watclp2,10,dist,105)
/*
/* setwindow cost_grid1
/* cost_dist1 = costdistance(start_cels,cost_grid1,back_link1,al_grid1,#,#)
/* setwindow cost_grid2
/* cost_dist2 = costdistance(start_cels,cost_grid2,back_link2,al_grid2,#,#)
/* setwindow cost_grid1
/* cost_path1 = costpath(ends,cost_grid1,back_link1,bycel)
/* setwindow cost_grid2
/* cost_path2 = costpath(ends,cost_grid2,back_link2,bycel)
/* setwindow cost_path1
/* lake_path1 = gridline(cost_path1,data,nothin,nofilter,sharp,#,#,#,15)
/* setwindow cost_path2
/* lake_path2 = gridline(cost_path2,data,nothin,nofilter,sharp,#,#,#,15)
&ty Step 5, run the costdistance calculation to develop the cost_distance
/* from startcells i.e lake labels to other cels
```

```
cost_dist = costdistance(start_cels,cost_grid,back_link,al_grid,#,#)
&ty Step 6, run the costpath analysis to develope the shortest path between
/* end points where stream enter and leave lakes and the lakes label point
cost_path = costpath(ends,cost_grid,back_link,bycel)
&ty Step 7, create the inside lake vectors from the cost_path analysis
lake_path = gridline(cost_path,data,nothin,nofilter,sharp,#,#,#,15)
quit
&ty Step 8, append together the inside lake_paths with the streams network, water_cl
copy water_cl stream_seg
dropitem stream_seg.aat stream_seg.aat mapid id lake_bnd
build stream_seg line
dropitem stream_seg.aat stream_seg.aat lake_bnd
append joined_lines line all
lake_path
stream_seg
end
/* If buff_watclp was split into two parts
/* append joined_lines line all
/* lake_path1
/* lake_path2
/* end
build joined_lines line
&ty Step 9, cleaning line work to snap to input and output streams intersection points
matchnode joined_lines matchnode5 5 lake_ends extend
build matchnode5 line
matchnode matchnode5 matchnode10 10 lake_ends extend
build matchnode10 line
matchnode matchnode10 matchnode15 15 lake_ends extend
build matchnode15 line
matchnode matchnode15 matchnode20 20 lake_ends extend
build matchnode20 line
matchnode matchnode20 matchnode25 25 lake_ends extend
build matchnode25 line
/* final coverage is w_arcs_cl
rename matchnode25 w_arcs_cl
build w_arcs_cl line
arcedit
editcov w_arcs_cl
editf arc
sel all
unsplit
save
quit
build w_arcs_cl line
&type killing off temp files
/*  &call kill2
&type Have finished creating the through lake part of the stream network
&watch &off
&return
&routine kill
&ty         Killing temp covers at start of aml
/* Kill Coverages
```

```
&if [exists lgstartbuffs -cover] &then &do
kill lgstartbuffs all
&end
&if [exists lg_st_buf_cl -cover] &then &do
kill lg_st_buf_cl all
&end
&if [exists lglstarts -cover] &then &do
kill lglstarts all
&end
&if [exists buff_watline -cover] &then &do
kill buff_watline all
&end
&if [exists islands_buff -cover] &then &do
kill islands_buff all
&end
&if [exists buffers -cover] &then &do
kill buffers all
&end
&if [exists buff_watclp -cover] &then &do
kill buff_watclp all
&end
&if [exists w_arcs_cl -cover] &then &do
kill w_arcs_cl all
&end
&if [exists joined_lines -cover] &then &do
kill joined_lines all
&end
&if [exists stream_seg -cover] &then &do
kill stream_seg all
&end
&if [exists sel_lake_buf -cover] &then &do
kill sel_lake_buf all
&end
&if [exists lake_path -cover] &then &do
kill lake_path all
&end
&if [exists water_lclp -cover] &then &do
kill water_lclp all
&end
&if [exists matchnode5 -cover] &then &do
kill matchnode5 all
&end
&if [exists matchnode10 -cover] &then &do
kill matchnode10 all
&end
&if [exists matchnode15 -cover] &then &do
kill matchnode15 all
&end
&if [exists matchnode20 -cover] &then &do
kill matchnode20 all
&end
&if [exists matchnode25 -cover] &then &do
```

```
kill matchnode25 all
&end
/* Kill Grids
&ty         Killing temp grids at start of aml
&if [exists mask -grid] &then &do
kill mask all
&end
&if [exists  starts -grid] &then &do
kill starts all
&end
&if [exists start_cels -grid] &then &do
kill start_cels all
&end
&if [exists ends -grid] &then &do
kill ends all
&end
&if [exists cost_grid -grid] &then &do
kill cost_grid all
&end
&if [exists cost_dist -grid] &then &do
kill cost_dist all
&end
&if [exists  back_link -grid] &then &do
kill back_link all
&end
&if [exists al_grid -grid] &then &do
kill al_grid all
&end
&if [exists cost_path -grid] &then &do
kill cost_path all
&end
&if [exists lake_path -grid] &then &do
kill lake_path all
&end
&return
/* The folowing routine creates extra paths through the lake to improve connectivity
&routine buffer
&if [exists big_lakes -cover] &then &do
kill big_lakes all
&end
arcedit
editcov sel_lake_dis
editf poly
asel all
reselect area > 2400000
put big_lakes
asel all
editf label
asel all
reselect area > 2400000
put lglstarts
quit
```

```
/* loop through following code if there are lakes > 240 hectares
&if [exists big_lakes -cover] &then &do
&do buffname &list 200 400 600 800 1000 1200 1400 1600 1800 2000 2200 2400
/*  2600 2800 3000 5000 7000 10000 15000 20000 30000
&if [exists buff%buffname% -cover] &then &do
kill buff%buffname% al
&end
&end
build big_lakes poly
buffer big_lakes buff200 # # -200 0.001 poly # #
buffer big_lakes buff400 # # -400 0.001 poly # #
buffer big_lakes buff600 # # -600 0.001 poly # #
buffer big_lakes buff800 # # -800 0.001 poly # #
buffer big_lakes buff1000 # # -1000 0.001 poly # #
buffer big_lakes buff1200 # # -1200 0.001 poly # #
/* buffer big_lakes buff1400 # # -1400 0.001 poly # #
/* buffer big_lakes buff1600 # # -1600 0.001 poly # #
/* buffer big_lakes buff1800 # # -1800 0.001 poly # #
/* buffer big_lakes buff2000 # # -2000 0.001 poly # #
/* buffer big_lakes buff2200 # # -2200 0.001 poly # #
/* buffer big_lakes buff2400 # # -2400 0.001 poly # #
/* buffer big_lakes buff2600 # # -2600 0.001 poly # #
/* buffer big_lakes buff2800 # # -2800 0.001 poly # #
/* buffer big_lakes buff3000 # # -3000 0.001 poly # #
/* buffer big_lakes buff5000 # # -5000 0.001 poly # #
/* buffer big_lakes buff7000 # # -7000 0.001 poly # #
/* buffer big_lakes buff10000 # # -10000 0.001 poly # #
/* buffer big_lakes buff15000 # # -15000 0.001 poly # #
/* buffer big_lakes buff20000 # # -20000 0.001 poly # #
/* buffer big_lakes buff30000 # # -30000 0.001 poly # #
&do buffname &list 200 400 600 800 1000 1200 1400 1600 1800 2000 2200 2400
/*  2600 2800 3000  5000 7000 10000 15000 20000 30000
&if [exists buff%buffname% -cover] &then &do
build buff%buffname% line
&end
&end
append buffers line al
buff200
buff400
buff600
buff800
buff1000
buff1200
/* buff1400
/* buff1600
/* buff1800
/* buff2000
/* buff2200
/* buff2400
/* buff2600
/* buff2800
/* buff3000
/* buff5000
/* buff7000
/* buff10000
/* buff15000
/* buff20000
/* buff30000
end
build buffers line
dropitem water_lclp.aat water_lclp.aat value
/* Lets buffer the start point so that it intersects other lines
&do buffnameb &list 100 200 300 400 500 600 700 800 900 1000 1200 1400 1600
1800
2000
&if [exists sbuff%buffnameb% -cover] &then &do
kill sbuff%buffnameb% all
&end
&end
buffer lglstarts sbuff100 # # 100 0.001 point # #
buffer lglstarts sbuff200 # # 200 0.001 point # #
buffer lglstarts sbuff300 # # 300 0.001 point # #
buffer lglstarts sbuff400 # # 400 0.001 point # #
buffer lglstarts sbuff500 # # 500 0.001 point # #
buffer lglstarts sbuff600 # # 600 0.001 point # #
buffer lglstarts sbuff700 # # 700 0.001 point # #
buffer lglstarts sbuff800 # # 800 0.001 point # #
buffer lglstarts sbuff900 # # 900 0.001 point # #
buffer lglstarts sbuff1000 # # 1000 0.001 point # #
buffer lglstarts sbuff1200 # # 1200 0.001 point # #
buffer lglstarts sbuff1400 # # 1400 0.001 point # #
buffer lglstarts sbuff1600 # # 1600 0.001 point # #
buffer lglstarts sbuff1800 # # 1800 0.001 point # #
buffer lglstarts sbuff2000 # # 2000 0.001 point # #
&do buffnameb &list 100 200 300 400 500 600 700 800 900 1000 1200 1400 1600
1800
2000
&if [exists sbuff%buffnameb% -cover] &then &do
build sbuff%buffnameb% line
&end
&end
append lgstartbuffs line all
sbuff100
sbuff200
sbuff300
sbuff400
sbuff500
sbuff600
sbuff700
sbuff800
sbuff900
sbuff1000
sbuff1200
sbuff1400
```

```
sbuff1600
sbuff1800
sbuff2000
end
build lgstartbuffs line
clip lgstartbuffs buff200 lg_st_buf_cl line 0.001
/* append together the clipped grid thinned lake lines, start cel buffers and the inside
buffer
lakes
append buff_watline line all
lg_st_buf_cl
water_lclp
buffers
end
/* Cut buffered Islands into line data set to cover off the width of line distance function
buffer islands islands_buff # # 20 0.001 poly # #
build islands_buff poly
erase buff_watline islands_buff buff_watclp line 0.001
clean buff_watclp buff_watclp 0.001 0.001 line
build buff_watclp line
/* kill off temp buffer files
&do buffname &list 200 400 600 800 1000 1200
/*  1400 1600 1800 2000 2200 2400
/*  2600 2800 3000 5000 7000 10000 15000 20000 30000
kill buff%buffname% all
&end
/* Lets kill the start point buffer
&do buffnameb &list 100 200 300 400 500 600 700 800 900 1000 1200 1400 1600
1800
2000
kill sbuff%buffnameb% all
&end
/* &end for if big_lakes exists
&end
/* If no big lakes exists
&else
copy water_lclp buff_watclp
&return
/* Data cleanup area
&routine kill2
&ty       Killing temp covers at end of aml
/* Kill Coverages
&if [exists lgstartbuffs -cover] &then &do
kill lgstartbuffs all
&end
&if [exists lg_st_buf_cl -cover] &then &do
kill lg_st_buf_cl all
&end
&if [exists lglstarts -cover] &then &do
kill lglstarts all
&end
&if [exists buff_watline -cover] &then &do
```

```
kill buff_watline all
&end
&if [exists islands_buff -cover] &then &do
kill islands_buff all
&end
&if [exists buffers -cover] &then &do
kill buffers all
&end
&if [exists buff_watclp -cover] &then &do
kill buff_watclp all
&end
&if [exists joined_lines -cover] &then &do
kill joined_lines all
&end
&if [exists stream_seg -cover] &then &do
kill stream_seg all
&end
&if [exists sel_lake_buf -cover] &then &do
kill sel_lake_buf all
&end
&if [exists lake_path -cover] &then &do
kill lake_path all
&end
&if [exists water_lclp -cover] &then &do
kill water_lclp all
&end
&if [exists water_cl -cover] &then &do
kill water_cl all
&end
&if [exists lake_labels -cover] &then &do
kill lake_labels all
&end
&if [exists lake_ends -cover] &then &do
kill lake_ends all
&end
&if [exists matchnode5 -cover] &then &do
kill matchnode5 all
&end
&if [exists matchnode10 -cover] &then &do
kill matchnode10 all
&end
&if [exists matchnode15 -cover] &then &do
kill matchnode15 all
&end
&if [exists matchnode20 -cover] &then &do
kill matchnode20 all
&end
/* Kill Grids
&ty      Killing temp grids at end of aml
&if [exists mask -grid] &then &do
kill mask all
&end
```

```
&if [exists  starts -grid] &then &do
kill starts all
&end
&if [exists start_cels -grid] &then &do
kill start_cels all
&end
&if [exists ends -grid] &then &do
kill ends all
&end
&if [exists cost_grid -grid] &then &do
kill cost_grid all
&end
&if [exists cost_dist -grid] &then &do
kill cost_dist all
&end
&if [exists  back_link -grid] &then &do
kill back_link all
&end
&if [exists all_grid -grid] &then &do
kill all_grid all
&end
&if [exists cost_path -grid] &then &do
kill cost_path all
&end
&if [exists lake_path -grid] &then &do
kill lake_path all
&end
&return
```