Route Building

Submitted By: Justin Deoliveira, Developer
Refractions Research Inc.
400 - 1207 Douglas Street
Victoria, BC, V8W-2E7
jdeolive@refractions.net
Phone: (250) 383-3022
Fax: (250) 383-2140

Table of Contents

Table of Figures

1 INTRODUCTION

A *stream network* provides the facility to process information with respect to an entire stream system, instead of to a single *stream segment*. A stream network alone however does not provide much information. Higher level constructs must be built on top of the stream network in order to provide useful information.

For instance, a *stream* is a structure that is composed of a contiguous collection of stream segments. The stream network itself can only tell us what set of segments are contiguous, and can be joined to form a stream. The decision of what segments actually form a single stream must be made by analyzing the relationships in the stream network.

Routing is the process of analyzing the stream network to decide which segments should be grouped into a set of streams. The term *route* is used to describe a collection of contiguous segments. Note that every stream is a route, but not every route is a stream. The route building process involves the construction of routes that do not end up in the final set of streams.

2 DEFINITIONS

Before getting to the specifics of the route-building algorithm, some definitions are required.

A *stream segment* is a continuous portion of stream that contains no breaks. Breaks occur when a stream segment meets another stream segment, or when the stream segment contains a source or a sink (see definitions below). The following figure contains three stream segments.

A *stream network* is a collection of stream segments and the associated relationships between the segments. A stream network is made up of *edges* and *nodes* (see definitions below).

An *edge* is the network representation of a stream segment. Since an edge directly references a single stream segment, the two terms are used synonymously. An edge contains exactly two *nodes* (see definition below) and can be *directional* or *un-directional*. A *directional* edge represents a stream segment that flows from one node to another. An *un-directional* edge represents a stream segment for which the direction of flow is either unknown, or bi-directional.

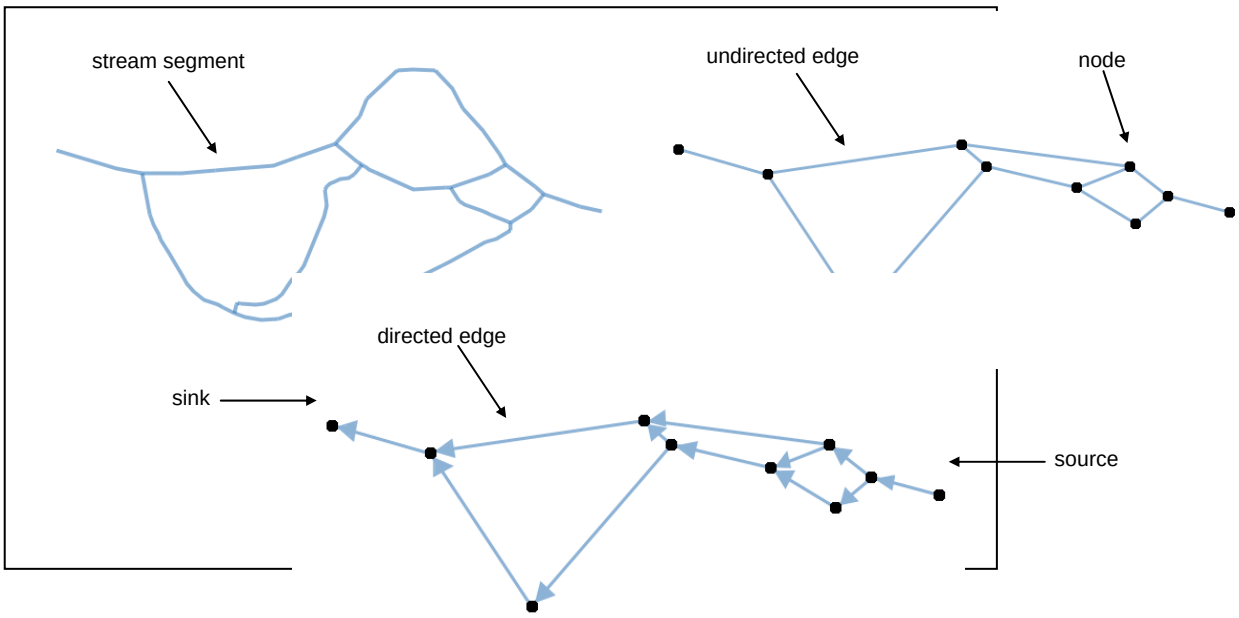
A *node* is a point of connectivity between two edges. A node is said to be *adjacent* to an edge if that edge contains the node. A node can be adjacent to any number of edges.

A *source* is a node in which every adjacent node flows from it to the other node contained by the edge.

A *sink* is a node in which every adjacent node flows to it from the other node contained by the edge.

An example of a stream system is shown on the top left of the following figure; the associated undirected stream network on the top right; and the associated directed stream network on the bottom.

Figure 1: Stream Segments



A *route* is the representation of a network flow. It is made up one or more contiguous edges. A route contains an *edge set* and a *node sequence*. The direction of the route is inferred from the order of the edge set and node sequence. Note that this direction may disagree with the direction of an individual edge in the edge set.

A *path* is a sequence of nodes and edges, similar to a route but without any of the restrictions. A path is used to define a pre-route structure. As previously described, all routes are paths, but not all paths are routes.

The *edge set* E of a route is the ordered set of edges contained by the route such that two consecutive edges in the set share a node. Note that classification as a

set implies that the edge set does not contain duplicate edges. The size of the edge set is always greater than zero and one less than the size of the edge set.

The *node sequence* N of a route is the sequence of nodes contained by the route such that two consecutive nodes in the sequence share an edge. An additional constraint is placed on the node sequence such that all the nodes must be distinct except for the first and last. The size of the node sequence is always one greater than the size of the edge set.

The following is a mathematical definition of a route:

R = (**E**,**N**) such that |E| > 0 and |N| = |E| + 1
E = $\{e_1, e_2, \ldots, e_m\}$ such that m = |E| and ***nodes***$(e_i) \cap$ ***nodes***(e_{i+1}) != {} for i = 1 .. m-1
N = $\{n_1, n_2, \ldots, n_{m+1}\}$ such that ***edges***(n_i, n_{i+1}) != { } for i = 1 .. m and $n_j \neq n_k$
for all (j,k) such that j < k for j = 1 .. m, k = 2 .. m+1

nodes(e) = $\{n_1, n_2\}$ such that n_1 and n_2 are adjacent to e
edges(n1,n2) = $\{e_1, e_2, \ldots, e_k\}$ such that ***nodes***$(e_i) \cap$ {n1,n2} != {} for i = 1..k

The *end sequence* of a route is the two-element sequence that contains the first and last elements of the node set of the route.

The *internal set* of a route is the ordered set that contains all nodes in the node sequence except for the first and last. The internal set is equal to the node sequence minus the end sequence.

The *edge intersection* of two routes is the set of edges that are shared by the routes. More precisely, it is the set intersection of the edge sets of the respective routes.

The *node intersection* of two routes is the set of nodes that are shared by routes. More specifically, it is the set intersection of the node sets of the respective routes.

A *child route* of a route, R1, is any other route that flows out of R1. More precisely, a route, R2, is a child of R1 if the node intersection of the two routes contains the last node of R2. R1 is also known as the *parent route* of R2.

Two routes are in *conflict* if they share an intermediate node and flow through each other. In other words, two routes are in conflict if the intersection of the internal sequences of the respective routes is not empty.

A *route set* is a set of routes. A route set is *proper* if it obeys the following constraints.

1. There are no route conflicts. In other words, no edges are shared and the intersection of the edge sets of every route must be the empty set.

2. Two routes can share up to a maximum of two nodes only if all the shared nodes belong to the end sequence of either route. As such, the node intersection of two routes must be a subset of the union of the end sequences of the respective routes.

The following is a mathematical definition of a route set:

```
RS = {Ri} such that:

1. edgeIntersection(Ri,Rj) = {} for all i,j = 1 .. |RS| and i != j
2. nodeIntersection(Ri,Rj) subset of endNodes(Ri) union endNodes(Rj) for all i,j = 1 ..
|RS| and i != j

R1 = (E1,N1)
R2 = (E2,N2)
edgeIntersection(R1,R2) = {e | e ∈ E1 and e ∈ E2}
nodeIntersection(R1,R2) = {n | n ∈ N1 and n ∈ N2}
endNodes(R1) = {n1} union {nm} such that m = |N1|
```

Figure 2: Improper Route Set

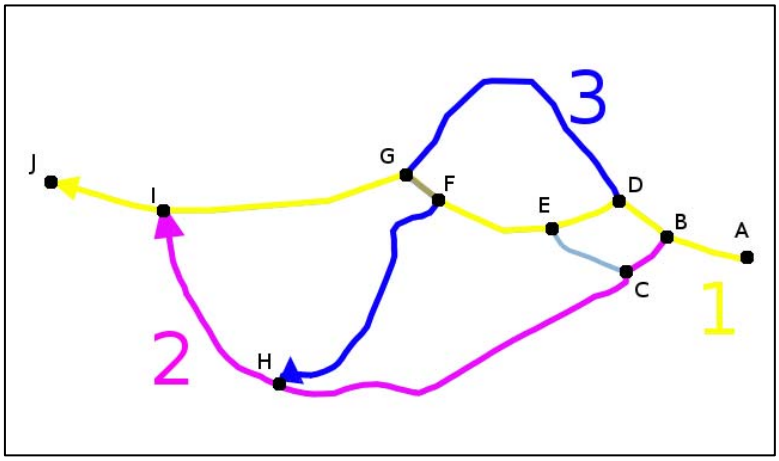


Figure 2 shows an improper route set. The shared edge (F,G) causes a conflict between routes 1 and 3. Routes 2 and 3 are both children of route 1.

The properties of the above route set are summarized below.

```
Route 1
   node sequence = {A,B,D,E,F,G,I,J}
   edge set = {(A,B),(B,D),(D,E),(E,F),(F,G),(G,I),(I,J)}
   end sequence = {A,J}
   internal set = {B,D,E,F,G,I}

Route 2
   node sequence = {B,C,H,I}
   edge set = {(B,C),(C,H),(H,I)}
   end sequence = {B,I}
   internal set = {C,H}

Route 3
   node sequence = {D,G,F,H}
   edge set = {(D,G),(G,F),(F,H)}
   end sequence = {D,H}
```

```
   internal set = {G,F}

Routes 1 and 2
   node intersection = {B,I}
   edge intersection = {}

Routes 1 and 3
   node intersection = {D,G,F,H}
   edge intersection = {(F,G)}

Routes 2 and 3
   node intersection = {}
   edge intersection = {}
```

3 ROUTE-BUILDING ALGORITHM

3.1 Inputs

The route-building algorithm takes two types of inputs The first is a fully connected, properly noded stream network. The second is a set of *pre-defined routes*.

A *pre-defined route* is a route that is known to exist. Specifying the pre-defined route set serves two purposes. First, it forces certain routes to be created by the algorithm, and second it serves as a seed for the algorithm.

3.2 The Algorithm

The following steps outline the major components of the route-building algorithm:

1. Create a set of final routes, which is initially the pre-route set.
2. For each route in the final set, create a set of child paths and add them to a candidate set.
3. Filter the candidate set of paths down removing any paths that form illegal routes.
4. Sort the set of candidate paths and remove any conflicting paths.
5. Create a route from each path and add it to the pre-route set.
6. Return to step 2 if the remaining candidate set is not empty.

The following is a pseudo-code description of the algorithm.

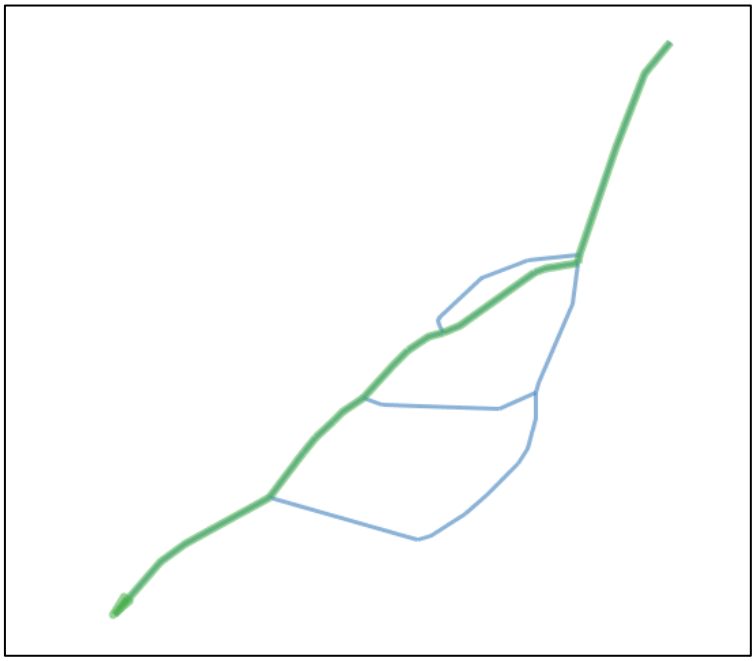
```
R = set of pre-routes
do
     P = empty set
     for each pre-route r in R
           C = set of child routes of r
           C_filtered = filter C
           P = P union C_filtered
     end for

     P_sorted = sort P
     P_sorted = P_sorted – set of conflicts

     for each Path p in P
           r = create route from p
           R = R union {r}
while P_sorted not empty
```

Shown in Figure 3 is a stream network for which a single pre-defined route has been specified (in green).

Figure 3: Stream Network with a Single Pre-defined Route



In the second step of the algorithm, we use the single pre-defined route to build child routes, which will then serve as candidate routes for the final sets. To calculate child routes, the following strategy is used.

```
R = parent route
C = {} //set of child routes

for each Node n in node set of R
     P = {} //set of paths
     P = P union {paths starting at n}
     for each Path p in P
           r = route from p
           C = C union {r};
```

A variety of path algorithms exist to find paths from a single node. Different algorithms create different paths, which will ultimately result in a different set of final routes. Three such algorithms will be discussed in the next section.

1. Exhaustive
2. Shortest path to sink
3. Path of least deflection

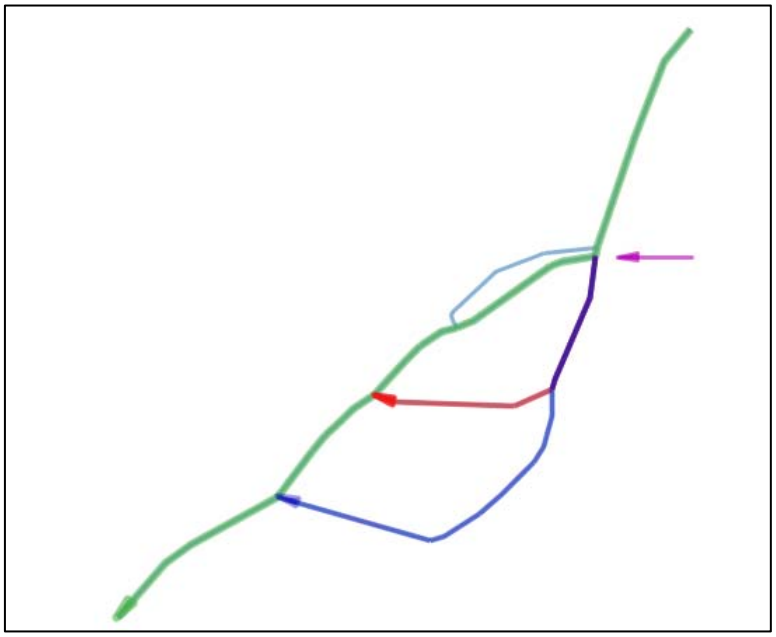
In order to build legal routes, a path from a node n must end on a node (not the same node n) that is part of another route, or is a sink node.

4 PATH DETERMINATION METHODS

4.1.1 Exhaustive Path Method

This method of path determination essentially computes all possible paths. In the following figure, an exhaustive path strategy from the specified node will result in two paths (red and blue) being created. These two paths have been calculated from the node specified by the purple arrow.

Figure 4: Result of Path Determination

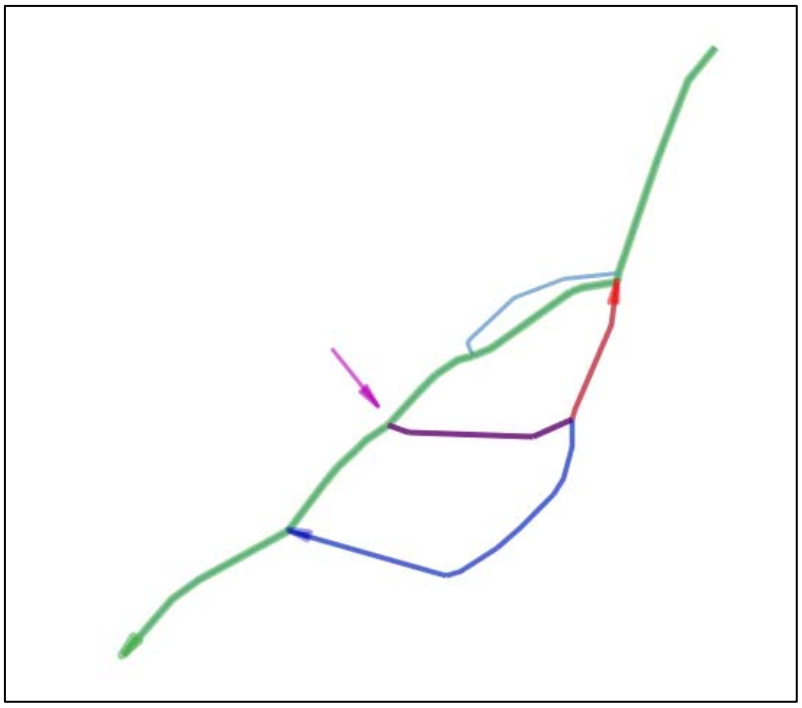


Apart from being the simplest method of path determination, the exhaustive method has little benefits. The first major problem is that calculating the

exhaustive set is expensive. The algorithm is exponential and becomes impractical for large networks.

A second downfall of the algorithm is that it produces unlikely paths. In the above figure this is not evident but consider the following figure in which the algorithm has been executed from a different node. Two paths (red and blue) have been calculated from the node specified by the purple arrow. The red path is an unlikely water course.

Figure 5: Unlikely Result of Path Determination



In the above figure, the red path is both unlikely and illegal. It is unlikely because the path turns back on the overall flow of the stream network. It is illegal because it causes a cycle in the routed network. For these reasons, exhaustive path determination is usually not used.

4.1.2 Shortest Path to Sink Method

The term “shortest path” is somewhat ambiguous. Most of the time it refers to the cumulative length of all the stream segments in the path; however, other weights besides length can be used. An example is channel width if the path determination is occurring inside of a water body. In this section, shortest path refers to length.

This method calculates paths based on a sink in the stream network. Paths from the sink node to every other node in the network are pre-calculated. When calculating the shortest path from a particular node, the following steps are followed. We refer to the *current* node as the node at which path determination is occurring.

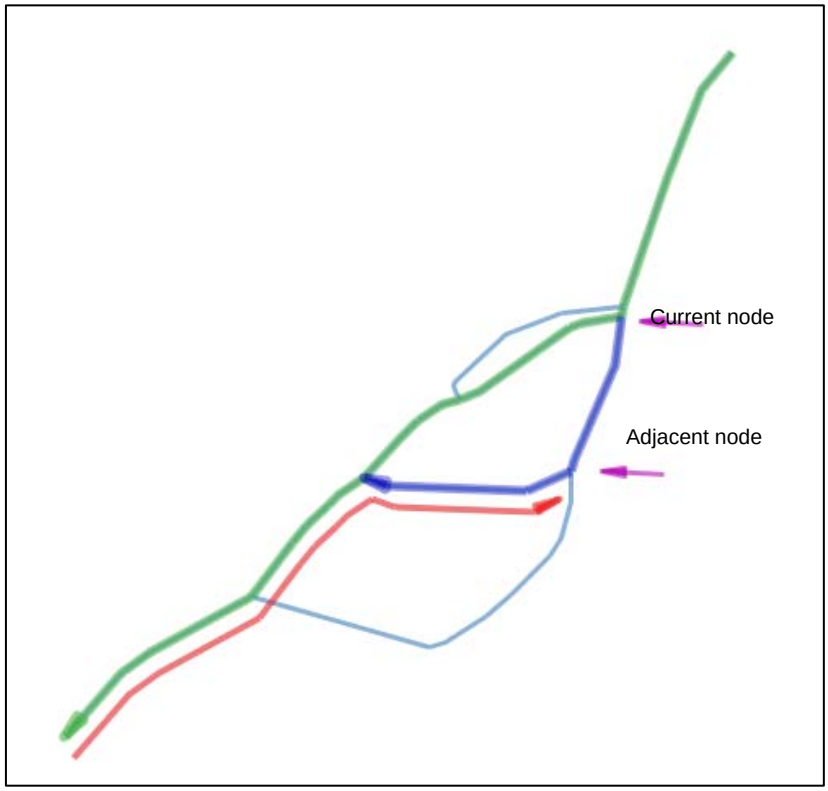
1. Find an adjacent node to the current node, which is not part of another route.
2. Determine the shortest path from the adjacent node to the sink node.
3. Append the current node to the beginning of the path.
4. Reject the path if it is invalid.

Firstly, the adjacent nodes to the current node are examined in search for one that does not belong to another route. If such a node cannot be found, path determination at the current terminates.

The next step uses the pre-calculated paths from the sink node to find the shortest path from the adjacent node to the sink, which is just the reverse of the path from the sink node to the adjacent node. The path is then truncated if it intersects another route. The current node is then pre-pended to the beginning of the path.

The following figure shows the shortest path from the sink node to the adjacent node and the corresponding path from the current node. The pre-calculated path to the adjacent node is shown in red. The resulting path from the current node is shown in blue.

Figure 6: Shortest Path Determination



It is possible for the shortest path from the adjacent node to the sink node to already contain the current node. In this case, pre-pending the current node creates a path that represents an invalid route. In this case, the path is rejected and path determination terminates.

This method of path determination addresses the issues with the exhaustive method presented in the preceding section. First off, the shortest path method is efficient. At first glance this may not seem so as the algorithm calculates the path to every single node in the network, some of which are never used; however, there exist incremental algorithms, which perform this calculation efficiently. The most notable is the infamous, *Dikkstra's Shortest Path Algorithm*.

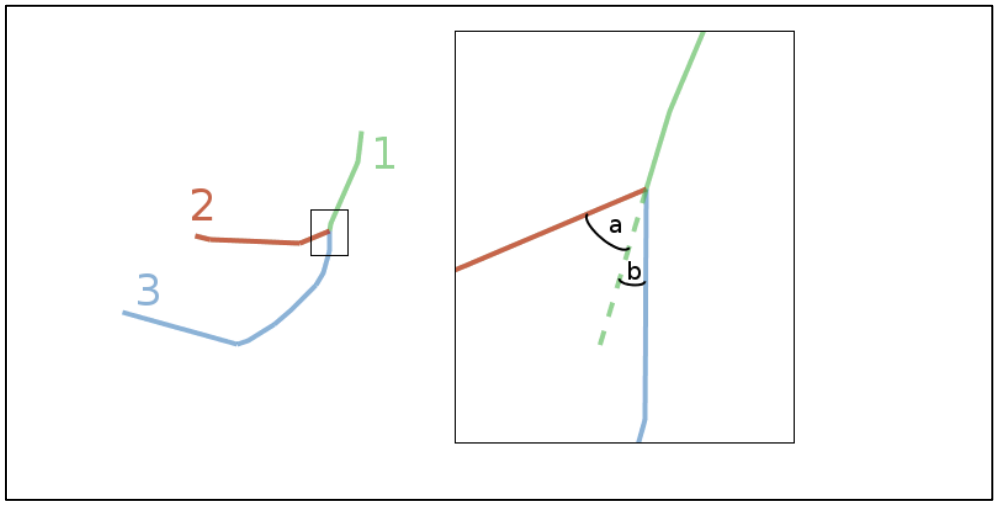
Furthermore, using shortest paths eliminates many of the undesired paths calculated with the exhaustive method.

This being said, the algorithm does have a few notable issues. The first is how to handle networks with multiple sinks. One solution is to make a choice which sink to use before the path is calculated. A second solution is to calculate the shortest path to every sink and have the sorting step of the overall algorithm (presented later) choose between them.

4.1.3 Path of Least Deflection Method

The term *deflection* refers to the angle made between two adjacent stream segments. Deflection is relative to a single segment. The following figure shows two deflection angles. The first is the deflection angle (a) of segment 2 relative to segment 1. The second is the deflection angle (b) of segment 3, relative to segment 1. In this figure, angle **b** is less than angle **a**. We can say that relative to segment 1, segment 3 deflects less than segment 2.

Figure 7: Two Deflection Angles



The deflection angle calculation is used to form a path from the current node to adifferent node that is contained by a route. At each step of the algorithm the path is extended by one node.

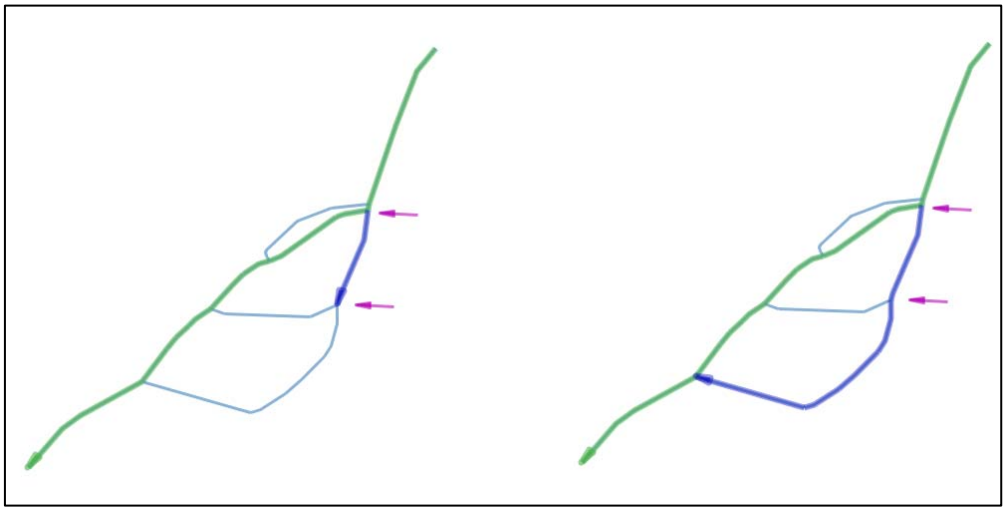
Consider an intermediate stage of the algorithm. The next node to be added to our path is chosen by examining the deflection angle of each edge adjacent to the last node currently in the path. The node that is incident with the chosen edge (there can only be one choice since the last node in currently in the path is also incident to the edge) is added and the algorithm continues.

The algorithm terminates when a node that is part of another route has been added to the path.

As with the shortest path method, the algorithm uses the current node and an adjacent node to seed the path with a single edge.

The following figures show the steps taken by the algorithm. On the left is the first step of the algorithm in which the first edge is chosen. On the right is the second and final step in which the next node is chosen from the deflection angle.

Figure 8: Path of Least Deflection



5 PATH REJECTION AND CONFLICT RESOLUTION

At this stage of the algorithm, path determination has been completed. The next step is to remove any paths that will lead to undesirable or illegal routes. Depending on the application and the data, different business rules for rejecting routes can be applied at this stage of the algorithm.

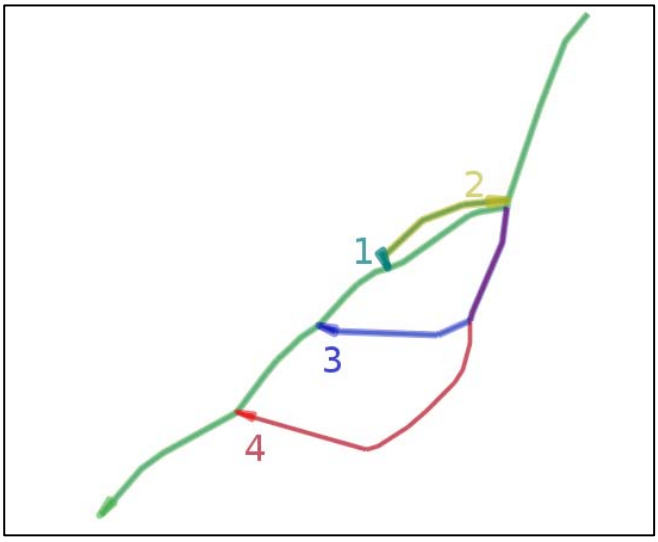
A number of criteria can be applied to reject routes; however, there are certain criteria that are common to many routing applications.

The first is *cycle creation*. Any path that will create a cycle in the routed network is rejected.

Another type of criteria involves the *direction* of the underlying stream network. Depending on the integrity of the stream dataset and the digitization rules, the underlying direction of the stream segments can provide a guide for path determination. Any path whose direction disagrees with any of the underlying segments should be rejected.

With these rules in mind consider the following figure, which depicts the algorithm after the path determination step. In this figure, four paths have been computed. In applying the rejection criteria described above, path 2 is rejected as it creates a cycle in the routed network.

Figure 9: The Algorithm State After Path Determination



Note that after the path rejection step there are still paths that are in conflict. The next step is to resolve these conflicts. This involves sorting the remaining paths by a specified criteria. As with path rejection criteria, path-sorting criteria varies with the application and data.

Examples of sorting criteria include:

- path length.
- path straightness (sum of deflection angles).
- length along parent route (the length along the parent route from the start node or the path, to the end node of the path).
- destination-flow angle (the angle made with the last edge of the path and the route the path flows into).
- source-flow angle (the angle made with the first edge of the path and the route the path flows out of).

Figure 10: Examples of Path Sorting Criteria