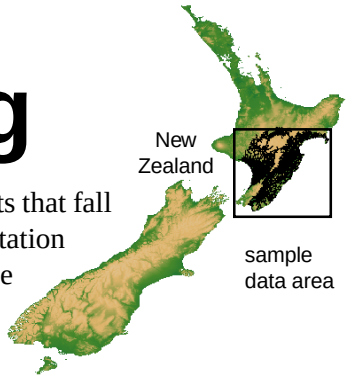
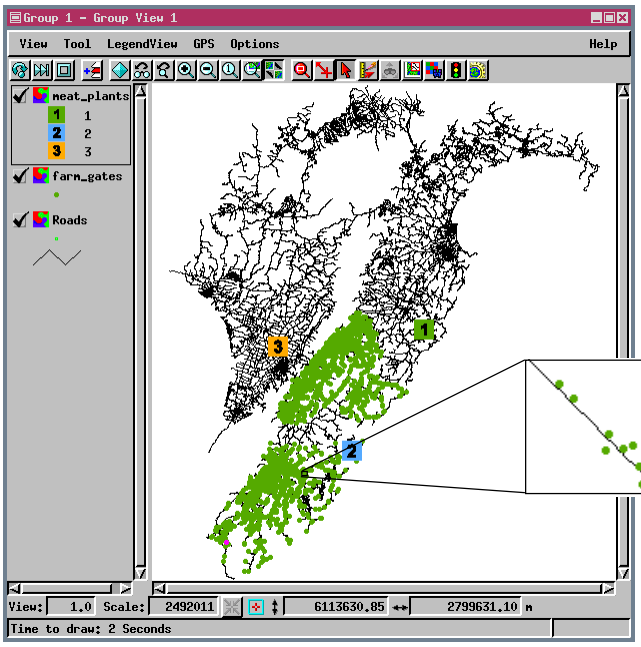
Sample SML Script

Farm to Market Routing



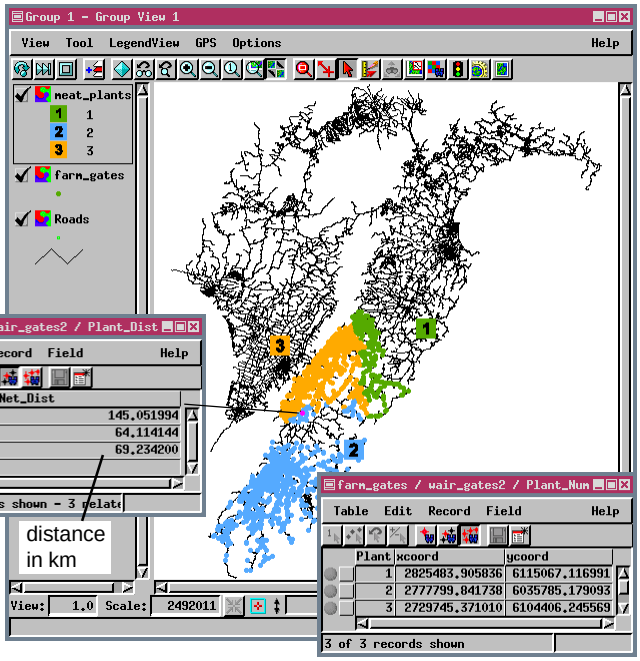
TNTmips includes a Network Analysis process that determines the "best" route between points that fall along a set of lines or the allocation of lines for the most efficient use in delivery to or transportation from a set of centers. In either case, the stops along the route or the centers must fall on the line network (the process automatically chooses the nearest node when you indicate the location of a stop or center). But what do you do when your points, in this case farm gates and processing plants, are not actually on the roads? Use an SML script like the one described here.



This script uses three vector objects: one to provide the road network, one with farm gate locations, and one with processing plant locations. You can substitute any widely distributed product location for the farm gates and any central location to which the product would be delivered for the processing plants. The difficulty in this particular case is that the data is not suitable for direct use in the Network Analysis process because, even if merged into a single vector object, the points do not fall on the roads (you may have to zoom in quite a way for it to be evident, see inset at left). The problem data was provided by AgriQuality New Zealand (formerly part of the Ministry of Agriculture and Forestry). This script uses only the distance from the processing plants to calculate impedance, but you can include a variety of other factors, such as road conditions, speed limits, and the price offered at each processing plant. You can readily change the market components of the impedance on a daily basis if need be with the end result of a dynamic appraisal of the best market for delivery of your product today.

The script adds a node to the ROADS object at the closest point on the closest line for each of the points in the FARMS object. It keeps track of these added nodes in an array that associates them with the correct farm. Nodes are similarly added for each of the processing plants. The shortest distance between each farm and processing plant is calculated using network analysis functions. This script adds two new tables to the point database of the FARMS vector object: one with records attached to each point that list the distance to each of the processing plants and another that provides the geographic coordinates of each processing plant in the same coordinate system used by the FARMS object. The ROADS vector ends up with many new nodes (equal to the number of farms and processing plants) that are not required for topology. These nodes can easily be removed by filtering the vector (use the Remove Excess Nodes filter) if desired.

The script attaches multiple records to each farm gate point—one for each processing plant. Such multiple attachments mean you cannot simply style by attribute because the first record attached to every point reports the distance to the first processing plant. In order to style each point according to which processing plant is closest or can be reached with the least impedance, you need to style by script using a script designed to evaluate all attached records. The script used to style the results shown here is included on the back of this page along with the script that determines the distance to each of the processing plants.



Sample scripts have been prepared to illustrate how you might use the features of TNTmips' Spatial Manipulation Language (SML). If possible, the full script is printed below for your quick perusal. When a script is too long to fit on one page, key sections are reproduced below. The sample script illustrated can be downloaded from the SML script exchange at www.microimages.com/sml/ftpsmllink/TNT_Products_V6.5_CD.

Script for Market Routes (network.sml)

```
clear();  #clear console

GetInputVector(Farms); #this vector is modified since tables are written to it
GetInputVector(Plants);
GetInputVector(TempNetwork); #this vector is modified by adding nodes

VectorToolkitInit(TempNetwork,"NoDBStatTable");
VectorToolkitInit(Farms);

numeric numPoints;
numeric numFarms;
numeric numPlants;
numeric numLines;
numeric i;

Array xarray[1];
Array yarray[1];

numFarms = NumVectorPoints(Farms);
Array farms[numFarms];
numeric linenumber;
numeric tempx;
numeric tempy;
numeric a;
numeric b;
numeric distance;

farmgeo = GetLastUsedGeorefObject(Farms);
tempgeo = GetLastUsedGeorefObject(TempNetwork);

printf("The number of farms is %d\n",numFarms);

for i=1 to numFarms {
        SetStatusMessage(sprintf("Processing point %d of %d of farms",i,numFarms));
        tempx = Farms.point[i].Internal.x;
        tempy = Farms.point[i].Internal.y;
        GeorefTrans(farmgeo,tempx,tempy,tempgeo,tempx,tempy);
        linenumber = FindClosestLine(TempNetwork,tempx,tempy);
        ClosestPointOnLine(TempNetwork,linenumber,tempx,tempy,a,b);
        VectorAddNode(TempNetwork,a,b,1);
        farms[i] = FindClosestNode(TempNetwork,a,b);
        }

numPlants = NumVectorPoints(Plants);
Array plants[numPlants];

plantgeo = GetLastUsedGeorefObject(Plants);

printf("The number of plants is %d\n",numPlants);

for i=1 to numPlants {
        SetStatusMessage(sprintf("Processing point %d of %d of plants",i,numPlants));
        tempx = Plants.point[i].Internal.x;
        tempy = Plants.point[i].Internal.y;
        GeorefTrans(plantgeo,tempx,tempy,tempgeo,tempx,tempy);
        linenumber = FindClosestLine(TempNetwork,tempx,tempy);
        ClosestPointOnLine(TempNetwork,linenumber,tempx,tempy,a,b);
        VectorAddNode(TempNetwork,a,b,1);
        plants[i] = FindClosestNode(TempNetwork,a,b);
        }

VectorUpdateStdAttributes(TempNetwork);
CloseVector(TempNetwork);  #flush vector

class Network net;
class Route route;
class MultiRoute multiroute;
numeric imp;

net =
NetworkInit(GetObjectFileName(TempNetwork),GetObjectName(GetObjectFileName(TempNetwork),GetObjectNumber(TempNetwork)));
NetworkSetDefaultAttributes(net);

numLines = NumVectorLines(TempNetwork);
for i=1 to numLines {
        imp = (TempNetwork.line[i].LINESTATS.Length);
        NetworkLineSetImpedance(net,i,imp,"FromTo");
        NetworkLineSetImpedance(net,i,imp,"ToFrom");
        }

total = numFarms * numPlants;
numeric count;
count = 1;
string tablename$;
class DATABASE db;
class DBTABLEINFO tinfo;

db = OpenVectorPointDatabase(Farms);
numeric recordnumber;
Array records[1];
numeric distance;

tinfo = TableCreate(db,"Plant_Num","Created by SML script");
TableAddFieldInteger(tinfo,"Plant",3);
TableAddFieldFloat(tinfo,"xcoord",25,6);
TableAddFieldFloat(tinfo,"ycoord",25,6);
for j=1 to numPlants {
        tempx = Plants.point[j].Internal.x;
        tempy = Plants.point[j].Internal.y;
        GeorefTrans(plantgeo,tempx,tempy,tempgeo,tempx,tempy);
        recordnumber = TableNewRecord(tinfo,j,tempx,tempy);
        records[1] = recordnumber;
        TableWriteAttachment(tinfo,i,records,1);
        }

tablename$ = "Plant_Dist";
tinfo = TableCreate(db,tablename$,"Created by SML Script");
TableAddFieldInteger(tinfo,"Plant",3);
class DBFIELDINFO the_field;
the_field = TableAddFieldFloat(tinfo,"Net_Dist",25,6);
the_field.UnitType = "Distance";
the_field.Units = "kilometers";

for j=1 to numPlants {

        printf("Plant %d\n",j);
        SetStatusMessage(sprintf("Calculating all routes from plant %d of %d",j,numPlants));
        NetworkCalculateMultiRoute(net,plants[j],farms,numFarms,multiroute);

        for i=1 to numFarms {
                SetStatusMessage(sprintf("Calculating route %d of %d",count,total));
                count +=1;
                printf("Route from plant %d to farm %d\n",j,i);

                NetworkMultiRouteGetRoute(multiroute,farms[i],route);
                report$ = NetworkRouteGetReport(route);

                distance = StrToNum(GetToken(report$, " ",18));
                recordnumber = TableNewRecord(tinfo,j,distance/1000);  #m/1000 = km
                records[1] = recordnumber;
                TableWriteAttachment(tinfo,i,records,1);

                NetworkRouteClose(route);
                }
        NetworkMultiRouteClose(multiroute);
        }

NetworkClose(net);

CloseVector(Farms);
CloseVector(TempNetwork);
CloseVector(Plants);

printf("Script Ran to Completion");
```

prompts for input vectors

variable declarations

gets georeference for farms and roads

finds closest point on closest line to farm gate and adds a node

creates array of all added nodes and corresponding farms

finds closest point on closest line to processing plant and adds a node

updates standard attributes and closes modified vector object (road network)

sets network impedance to line length

creates table that reports plant locations in same coordinate system as farms

creates table and field with km distance units

calculates distance of best route from each farm to each processing plant and adds to record attached to farm in distance table

Script for Styling by Closest Plant

```
val = Plant_Dist[1].Net_Dist
id = Plant_Dist[1].Plant

for i = 2 to SetNum(Plant_Dist[*]) {
  if (Plant_Dist[i].Net_Dist < val) {
     val = Plant_Dist[i].Net_Dist;
     id = Plant_Dist[i].Plant;
     }
}
```

finds processing plant with lowest distance value

```
if (id == 1)
   Style$ = "Style1" else
if (id == 2)
   Style$ = "Style2" else
   Style$ = "Style3"
UseStyle = 1
```

assigns drawing style according to plant identified as closest