CostDistanceMatrixMultiGrid

```
' Name:  Friction.CostDistanceMatrixMultiGrid2.0
' Title: For a set of cost grids, calculation of the cost matrix among a set of points or
polygons
'
' Author: Nicolas Ray, Anthropology and Ecology Dpt, Genetics and Biometry Lab.,
'         University of Geneva, 1227 Carouge, Switzerland
'         Email: nicolas.ray@anthro.unige.ch
'         http://lgb.unige.ch/home/ray
'
' Date: 25/09/02
'
' Version: 2.0
'
' Topics: Cost distance, matrix, Spatial Analyst, batch
'
' Description: this script is not standalone. He is to be used with
"Friction.costdistancematrix2.0"
' it is doing a costdistance matrix as the latter script, but for a set a grids that the used
can choose on disk
'
' Requires: The Spatial Analyst extension to be loaded. Friction.costdistancematrix2.0'
'
' Note:
'
'
' Call: this script calls the script "Friction.costdistancematrix2.0"
'
' Returns: This script the output of Friction.costdistancematrix2.0 for each input grid
'
'
' %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

'Test for Spatial Analyst
test=Extension.Find("Spatial Analyst")
if (test=NIL) then
 msgbox.error("You must have the spatial analyst extension loaded","CostMatrix")
 return(nil)
end

'Describe how cost distance calculation will occur and allow cancellation
response=MsgBox.YesNo("A set of grid will be chosen in order to compute a cost distance matrix
among all points of a chosen polygone or point theme. Continue?","Continue?",false)
if (response=nil) then exit end

'Get the active view and check if enough themes exist to perform operation
theView = av.GetActiveDoc 'Uses the active view
themeList = theView.getthemes
if (nil = themeList) then exit end
if (themeList.count < 2) then
  msgbox.error("Need at least 2 themes in the View","Error")
  exit
end

'List the points and polygone themes
polylist = list.make
for each atheme in themelist
 if (atheme.canselect=true) then
   themeType = atheme.getftab.findfield("Shape").gettype
   if ((themeType = #FIELD_SHAPEPOLY) OR (themeType = #FIELD_SHAPEPOINT)) then
     polylist.add(atheme)
   end
 else
 end
end
if (polylist.Count=0) then
  msgbox.error("You need at least one point or one polygone theme in the view!","Error")
  exit
end

' Choice of the point theme
thePolyTheme = MsgBox.ChoiceAsString(polylist,"Choose your point or polygone theme","Point and
Polygone theme")
if (thePolyTheme=Nil) then exit end
```

```
'***
'asks the user to select the grids
gridFNlist = SourceManager.ReturnDataSets(GRID, "Grids to work on")
if (gridFNlist = NIL) then return NIL end
if (gridFNlist.Count < 1) then return NIL end 'End the program if there is no grids to combine

'Make a list for the grids
GridList = List.Make

'Get the grids for each file name and put them in gridlist
for each gridFN in gridFNList
  gridSrcName = Grid.MakeSrcName(gridFN.AsString)
  gridtoadd = Grid.Make(gridSrcName)
  'check each grid for problems
  if (gridtoadd.HasError) then
    MsgBox.Info("gridtoadd.HasError = TRUE", "ERROR")
    return NIL
  end
  gridtoadd2 = gtheme.make(gridtoadd)
  GridList = GridList.add(gridtoadd2)
end

'***

' Choice of the max distance for calculation
maxdistance = MsgBox.input("Enter the max distance for calculation. Leave blank for full
extent", "Maxdistance", " ")
if (maxdistance = " ") then maxdistance = NIL else maxdistance = maxdistance.asNumber end

' Choice of saving the cost distance grids
reponse = MsgBox.YesNo("Do you want to save each individual cost distance grid?"+NL+"(as
costG_1, costG_2, ... etc.)","Save?",false)

'set the current directory to the working directory
av.getProject.GetWorkDir.SetCWD

' Loop through the grids
for each gridtheme in gridlist
      gridname = gridtheme.GetName
      g = gridtheme.getgrid

    ' creation of the output table
      Gridfilename = gridname.AsFilename
      results = av.Run( "Friction.costdistancematrix2.0", { theView, g, thePolyTheme,
maxdistance, Gridfilename, reponse} )
end
```

CostDistanceMatrix

```
' Name:  Friction.CostDistanceMatrix2.0
' Title: Calculation of the cost matrix among a set of points or polygones
'
' Author: Nicolas Ray, Anthropology and Ecology Dpt, Genetics and Biometry Lab.,
'         University of Geneva, 1227 Carouge, Switzerland
'         Email: nicolas.ray@anthro.unige.ch
'         http://lgb.unige.ch/home/ray
'
' Date: 25/09/02
'
' Version: 2.0
'
' Topics: Cost distance, matrix, Spatial Analyst
'
' Description: With a cost grid, this script calculates the accumulative cost distance
' among all the points or polygones of a given
' theme. The script generates a .dbf table (matrix) that gives the accumulative cost among
' all the points.
' The maximum accumulative cost can also be taken into account, and the choice is given to
' save the accumulative
' cost grid for each point. The output matrix can be set triangular by settings the
' flag "_DiagonalMatrix_" to TRUE
```

```
' Requires: The Spatial Analyst extension to be loaded. The script also requires an active
' view with an active
' grid theme, which is the cost theme. The point or polygone theme must have an ID Field.
' This script will work if the view is projected, given that the cost grid has been
' previously reprojected in the View projection
'
' Note:  The cost distance grids which are created are of integer type, in order to avoid
' a large number of
' space-demanding floating grids to be created. This function can be easily modified
' by removing the ".int"
' at the end of the cost distance calculation. The matrix returned can be set diagonal
' or square by changing the
' flag _DiagonalMatrix_ at the beginning of the script
'
'
' Self:this script can be called, by a batch process, from the script
' "Friction.CostDistanceMatrixMultiGrid2.0"
'      if not called, it react standalone and allow the computation on one grid
'      selfView = SELF.Get( 0 )          'the current view
'      selfGrid = SELF.Get( 1 )          'the cost grid
'      selfthePolyTheme = SELF.Get( 2 )  'the polygone or poin theme
'      selfmaxDistance = SELF.Get( 3 )   'the max distance of computation
'      selfoutputTable = SELF.Get( 4 )   'the output table
'      selfreponse = SELF.Get( 5 )       'the choice of saving the costdistance grid
'
' Returns: This script return a matrix in a .dbf table (and in text mode if desired), and
' the accumulative cost grids if desired.
'
'
' %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

'SETTINGS//////////////////////////////////////////////////////////////////////////
'set this _DiagonalMatrix_ variable to TRUE if you want a diagonal output matrix
_DiagonalMatrix_=true
'set this _TextOutput_ variable to TRUE if you also want the cost matrix to be output in a
text file, without colum and line headers
_TextOutput_=true
'//////////////////////////////////////////////////////////////////////////////////

' check if the script has been called
_ScriptCalled = false
if (Self.GetClass.GetClassName="LIST") then

  ' if called from an other script, the following variables are set
  _ScriptCalled = true
  selfView = SELF.Get( 0 )
  selfGrid = SELF.Get( 1 )
  selfthePolyTheme = SELF.Get( 2 )
  selfmaxDistance = SELF.Get( 3 )
  selfoutputTable = SELF.Get( 4 )
  selfreponse = SELF.Get( 5 ) 'choice of saving the costdistance grid
end

if (_ScriptCalled) then

  theView = selfView
  theViewProjection=theView.GetProjection

  theActiveGrid=selfGrid
  theActiveGridCellSize = theActiveGrid.GetCellSize
  theActiveGridExtent = theActiveGrid.Getextent

  thePolyTheme = selfthePolyTheme
  themeType = thePolyTheme.getftab.findfield("Shape").gettype

  maxdistance = selfmaxDistance

  InputTable = selfoutputTable
  theVTab = Vtab.MakeNew(InputTable, dbase)
  OutputTable = Table.Make(theVTab)
  theVTab.StartEditingWithRecovery

  reponse = selfreponse

else
  'Test for Spatial Analyst
  test=Extension.Find("Spatial Analyst")
```

```
  if (test=NIL) then
   msgbox.error("You must have the spatial analyst extension loaded","CostMatrix")
   return(nil)
  end

  'Describe how cost distance calculation will occur and allow cancellation
  response=MsgBox.YesNo("The ACTIVE grid theme will be used to compute the cost distance
              matrix among all points of a chosen point theme. Continue?","Continue?",false)
  if (response= false) then exit end

  'Get the active view and check if enough themes exist to perform operation
  theView = av.GetActiveDoc
  theViewProjection=theView.GetProjection
  themeList = theView.getthemes
  if (nil = themeList) then exit end
  if (themeList.count < 2) then
    msgbox.error("Need at least 2 themes in the View","Error")
    exit
  end

  'Use the active grid theme, and get cell size and extent
  theActiveTheme=theView.GetActiveThemes.Get(0)
  if (theActiveTheme.Is(GTHEME) = false) then
    msgbox.error("The active theme must be a grid!","Error")
    exit
  end
  theActiveGrid=theActiveTheme.GetGrid
  theActiveGridCellSize = theActiveGrid.GetCellSize
  theActiveGridExtent = theActiveGrid.Getextent

  'List the points and polygone themes
  polylist = list.make
  for each atheme in themelist
   if (atheme.canselect=true) then
     themeType = atheme.getftab.findfield("Shape").gettype
     if ((themeType = #FIELD_SHAPEPOLY) OR (themeType = #FIELD_SHAPEPOINT)) then
       polylist.add(atheme)
     end
   else
   end
  end
  if (polylist.Count=0) then
    msgbox.error("You need at least one point or one polygone theme in the view!","Error")
    exit
  end

  ' Choice of the poly theme
  thePolyTheme = MsgBox.ChoiceAsString(polylist,"Choose your point or polygone theme","Point
                      and Polygone theme")
  if (thePolyTheme=Nil) then exit end

  ' Choice of the max distance for calculation
  maxdistance = MsgBox.input("Enter the max distance for calculation (in cost distance units).
Leave blank for full extent", "Maxdistance", " ")
  if (maxdistance = " ") then maxdistance = NIL else maxdistance = maxdistance.asNumber end

  ' Choice and creation of the output table
  InputTable = FileDialog.Put( "cost_table".asFileName, "*.*", "Enter the name of the output
table" )
  theVTab = Vtab.MakeNew(InputTable, dbase)
  OutputTable = Table.Make(theVTab)
  theVTab.StartEditingWithRecovery

  ' Choice of saving the cost distance grids
  reponse = MsgBox.YesNo("Do you want to save each individual cost distance grid?"+NL+"(as
              costG_1, costG_2, ... etc.)","Save?",false)

end  '_ScriptCalled

' Find number of object
thePolyFtab = thePolyTheme.GetFtab
ID = thePolyFtab.FindField("ID")
total=0
for each rec in thePolyFtab
  total = total+1
end
```

```
'set the current directory to the working directory
av.getProject.GetWorkDir.SetCWD

' Create an ID field
f1 = Field.Make("ID", #FIELD_CHAR, 10, 0)
theVTab.AddFields({f1})

' Add id to ID field
for each i in 0..(total-1)
  newrec = theVtab.addRecord
  IDvalue=thePolyFtab.ReturnValueNumber (ID, i)
  IDvalueName="ID"+IDvalue.asString
  theVtab.SetValue(f1, newrec, IDvalueName)
end

' Add IDi fields to the table
for each i in 1..total
  IDvalue=thePolyFtab.ReturnValueNumber (ID, i-1)
  f = Field.Make("ID"+IDvalue.asString, #Field_long, 15, 0)
  theVTab.AddFields({f})
end

' change the analysis environment, to be sure it's compliant with the extent of the input grid
ae = theView.GetExtension(AnalysisEnvironment)
ae.SetExtent(#ANALYSISENV_VALUE, theActiveGridExtent)
ae.SetCellSize(#ANALYSISENV_VALUE, theActiveGridCellSize)
ae.Activate

' create the table in text format if desired
if (_TextOutput_) then
   tf = TextFile.Make ((InputTable.asString+".dis").asFileName, #FILE_PERM_WRITE)
end

' Costdistance calculations
' ************************
IdField = thePolyFtab.Findfield("ID")
i=1

' loop through the points
for each rec in thePolyFtab

      ' convert to GRID
       thePolyFtab.GetSelection.ClearAll
       thePolyFtab.UpdateSelection
       thePolyFtab.getselection.Set(rec)
       thePolyFtab.UpdateSelection
       sourceGrid = Grid.MakeFromFTab (thePolyFtab, theViewProjection, IdField,
                       {theActiveGridCellSize, theActiveGridExtent})
       if (sourceGrid.haserror) then msgbox.error("Conversion to grid failed.","") exit end
       thePolyFtab.Getselection.ClearAll

      ' Costdistance calculation
      costTempGrid = (sourceGrid.CostDistance (theActiveGrid, Nil, Nil, maxdistance)).int

      ' check for errors in the cost distance grid
      if (costTempGrid.HasError) then
        MsgBox.Info("costTempGrid has errors", "ERROR")
        return NIL
      end

      ' set the output table field in which we will write
        IDvalue=thePolyFtab.ReturnValueNumber (ID, i-1)
        IDvalueName="ID"+IDvalue.asString
        tempField = theVTab.Findfield(IDvalueName)

        for each j in 0..(total-1)

          'treat the lower triangular part of the matrix, if required through _DiagonalMatrix_
          if (((j+1)>=i) OR (_DiagonalMatrix_=false)) then
            'treat the diagonal case faster
            if (i=(j+1)) then costDistanceMinimum=0 else
              if ((themeType = #FIELD_SHAPEPOLY)) then
                 '*******Treat the polygone case

                 ' extract grid by  polygone
                 thePolyFtab.GetSelection.ClearAll
                 thePolyFtab.UpdateSelection
```

```
                thePolyFtab.getselection.Set(j)
                thePolyFtab.UpdateSelection

                'Get bounds of clipping area as a rectangle
                thePolyThmExtent  = thePolyTheme.getselectedextent
                if (thePolyThmExtent .IsEmpty) then thePolyThmExtent =
                     thePolyTheme.ReturnExtent end

                'Get parameters for the new grid
                theFtab = thePolyTheme.GetFTab
                theProj = theView.GetProjection
                theCell = costTempGrid.GetCellSize
                theExtent = costTempGrid.GetExtent

                'the actual extraction occurs here
                tempGrid = Grid.MakeFromFtab(thePolyFtab,theProj,nil,{theCell,theExtent})
                ExtractedGrid = (tempGrid.IsNull).Con (tempGrid, costTempGrid)

                ' extract the lowest value, which is the cummulative cost value of the LCP
                ExtractedGridVTab = ExtractedGrid.GetVTab
                costDistanceMinimum = nil
                if (ExtractedGridVTab<>nil) then
                  GridValueField = ExtractedGridVTab.FindField ("Value")
                  costDistanceMinimum = ExtractedGridVTab.ReturnValueNumber (GridValueField,
                                        0)
                end
                '************************************
              else
                '*******Treat the point case

                ' extract grid by point
                thePolyFtab.GetSelection.ClearAll
                thePolyFtab.UpdateSelection
                thePolyFtab.getselection.Set(j)
                thePolyFtab.UpdateSelection

                ' extraction of the values under each point
                thePoint = thePolyFtab.GetLabelPoint (j)
                costDistanceMinimum = costTempGrid.CellValue (thePoint, theViewProjection)
                '************************************
              end
            end

            ' write result in table
            if (costDistanceMinimum<>nil) then
              theVTab.SetValueNumber(tempField, j, costDistanceMinimum)
            else
              theVTab.SetValueNumber(tempField, j, -1)
            end
          end
        end

      ' save the cost distance grids
      if (reponse) then
        if (costTempGrid.BuildVat = FALSE) then MsgBox.info("Cannot save cost grid", "ERROR")
         end
        costgridname = "costG_"++i.asString
        theFN = av.getproject.getworkdir.maketmp(costgridname,"")
        costTempGrid.SaveDataSet (theFN)
      end

      i=i+1
end

' save the table in text format if desired
Fields=theVTab.GetFields
NumField=Fields.Count
for each record in theVTab
  for each f in 1..(NumField-1) ' we skip the first record which is the label
    v = theVTab.ReturnValueString (Fields.Get(f), record)
    aString=v.asString+TAB
    tf.Write (aString, aString.Count)
  end
  tf.Write (NL, 1)
end

' save edits of the matrix of the table
```

```
saveEdits = TRUE
theVTab.StopEditingWithRecovery(saveEdits)

' close the text file if desired
if (_TextOutput_) then
   tf.Close
end
```