

CLUS101f.pas

[This program takes output from the HCPM clustering program and sets it up in records suitable for use with the FeedDist program.]

Global variables:

CLUfile1 CLUfile2 INfile BINfile do_batch areaname

CloseParam

Dataline

ClusterNumber

CloseWindow

SetRaster

WeightDensity

GlobalRasterSize

CmdParam

i

code

dens

(from globals.pas)

GR

coordinate_file

CR

batchfile

num_SAs

procedure ParseDataline

Local variables: c vstr code i

read values from **dataline** into the following variables:

ClusterNumber

GR.cx1[1]

GR.cx1[1] = GR.cx1[1]*0.001 GR.cy1[1]

GR.cy1[1] = GR.cy1[1]*0.001 for i = 2 to 4 do GR.cx1[i] = GR.cx1[1] GR.cy1[i] =

GR.cy1[1]next

GR.gTotalLines

GR.cx2[1] GR.cx2[1] = GR.cx2[1]*0.001

GR.cy2[1]

GR.cy2[1] = GR.cy2[1]*0.001

GR.cx2[2]

GR.cx2[2] = GR.cx2[2]*0.001

GR.cy2[2]

GR.cy2[2] = GR.cy2[2]*0.001

For i = 3 to 4

GR.cx2[i] = GR.cx2[1]

GR.cy2[i] = GR.cy2[1]

next

```
for i = 1 to 4
  GR.cx3[i] = GR.cx1[1]
  GR.cx4[i] = GR.cx1[1]
  GR.cy3[i] = GR.cy1[1]
  GR.cy4[i] = GR.cy1[1]
next GR.DepthToBedrock

GR.Hardness
GR.SoilTexture

GR.WaterTb

GR.MinSlope

GR.MaxSlope

end of procedure ParseDataLine
```

procedure rasterize

passed variables:
ClusterNumber

local constants:
big = 1.0e10

local variables:

x
y
ResLines
BusLines
NumberOfPoints
i
c
code
vstr
xtry
ytry
numtry
m
lix
lly
urx
ury
data2
indata
j
k
RasterSize
RasterTooLarge

NumberOfPoints = 0

```
for i = 1 to 8000
  x[i] = zero
  y[i] = zero
  ResLines[i] = zero
```

```

    BusLines[i] = zero
next
open file CLUfile2 and read header ('X, Y, Cluster')
open file infile and move to the 5th line in the file
loop until the end of either CLUfile2 or infile is reached
    read a line from CLUfile2 into the variable data2
    read a line from infile into the variable indata

    parse contents of data2 into the following variables (comma separated)
    xtry
    ytry
    numtry

    if numtry = ClusterNumber then
        NumberOfPoints = NumberOfPoints + 1
        x[NumberOfPoints] = xtry * 0.001
        y[NumberOfPoints] = ytry * 0.001

        read the 4th and 5th values in indata into the following variable (comma separated)
        ResLines[NumberOfPoints]
        BusLines[NumberOfPoints]

    end if
end loop

GR.gHouseholds = 0
GR.gBusinessLines = 0
GR.gPrivateLines = 0
GR.gSpecialLines = 0

llx = big
lly = big
urx = -big
ury = -big

for i = 1 to NumberOfPoints
    if x[i] < llx then llx = x[i]
    if y[i] < lly then lly = y[i]
    if x[i] > urx then urx = x[i]
    if y[i] > ury then ury = y[i]
    GR.gHouseholds = GR.gHouseholds + round(ResLines[i])
    GR.gBusinessLines = GR.gBusinessLines + round(BusLines[i])
next i

llx = llx - 0.5
lly = lly - 0.5
urx = urx + 0.5
ury = ury + 0.5

RasterSize = GlobalRasterSize

{ if raster size is not given externally, or it is too large, calculate it here. }

RasterTooLarge = false
if (max((urx - llx), (ury - lly)) / RasterSize > 50) then RasterTooLarge = true

```

```

if RasterTooLarge or (SetRaster = false) then
  RasterSize = max((urx - llx), (ury-lly)) / 50
End if

GR.MicroGridNS = RasterSize
GR.MicroGridEW = RasterSize
GR.nrow = round( max((urx-llx), (ury-lly)) / RasterSize)
GR.ncol = GR.nrow
GR.LowerLeftX = llx
GR.LowerLeftY = lly
GR.UpperRightX = urx
GR.UpperRightY = ury

m = (GR.cy1[1] - GR.SwitchY) / (GR.cx1[1] - GR.SwitchX)

if (m >= -1) and (m < 1) and (GR.cx1[1] >= GR.SwitchX) then
  GR.quadrant = 1
else if ((m >= 1) or (m < -1)) and (GR.cy1[1] >= GR.SwitchY) then
  GR.quadrant = 2
else if (m >= -1) and (m < 1) and (GR.cx1[1] < GR.SwitchX) then
  GR.quadrant = 3
else
  GR.quadrant = 4
end if

for i = 1 to GR.nrow
  for j = 1 to GR.ncol
    GR.Households[i,j] = 0
    GR.BusLines[i,j] = 0
    for k = 1 to NumberOfPoints
      if (x[k] >= llx + (j-1) * RasterSize)
        and (x[k] < llx + j * RasterSize)
        and (y[k] >= lly + (i-1) * RasterSize)
        and (y[k] < lly + i * RasterSize) then
        GR.Households[i,j] = GR.Households[i,j] + round(ResLines[k])
        GR.BusLines[i,j] = GR.BusLines[i,j] + round(BusLines[k])
      end if
    next k
  next j
next i

{Now calculate density, using area defined by convex hull of populated raster cells.}

GR.density = call dist_density
  passing variables:
  GR

end of procedure rasterize

procedure convert
passed variables:
xlon
ylat

```

```
reflon  
reflat  
xft  
yft
```

```
local constants:
```

```
EarthRadius_meters = 6367723  
FeetPerMeter = 3.28083989501312
```

```
local variables:
```

```
NSCirc  
EWCirc
```

```
NSCirc = 2 * pi * EarthRadius_meters * FeetPerMeter
```

```
EWCirc = NSCirc * cos(reflat * pi / 180)
```

```
yft = NSCirc * (ylat - reflat) / 360
```

```
xft = EWCirc * (xlon - reflon) / 360
```

```
end of procedure convert
```

```
procedure process
```

```
passed variables:  
areaname
```

```
local variables:
```

```
DoneWithClusters  
swx  
swy  
cenx  
ceny  
swxkf  
swykf  
c  
vstr  
code  
BAKFile  
TotalLines  
gtl
```

```
open file CLUFile1 with filename of {areaname}.CLU  
open file CLUFile2 with filename of {areaname}.CLU  
open file INFile with filename of {areaname}.IN  
create file BINfile with filename of {areaname}.BAK  
create file coordinate_file with filename of {areaname}.COO
```

```
read from file CLUfile1 until reach the value 'Cluster'
```

```
move to the third line of the file Infile
```

```
read line from Infile into variable dataline
```

```
parse values from dataline into the following variables: (comma separated)
```

```
skip first value
```

```
swx
swy
cenx
ceny
```

```
CR.OriginX = cenx
CR.OriginY = ceny
CR.Reference_latitude = ceny
```

```
write the values in CR to the file coordinate_file
```

```
call convert
passing:
swx
swy
cenx
ceny
*swxkf
*swykf
```

```
swxkf = swxkf * 0.001
swykf = swykf * 0.001
```

```
TotalLines = zero
```

```
DoneWithClusters = false
```

```
start loop
```

```
  read line from file CLUfile1 into dataline
```

```
  if dataline <> '' then
    create a new GR variable
    GR.SwitchX = swxkf
    GR.SwitchY = swykf
    call ParseDataline
```

```
    call rasterize
    Passing:
    ClusterNumber
```

```
    { This tells us which microgrid each customer location belongs in. }
```

```
    write values in the variable GR to the file BINfile
```

```
    TotalLines = TotalLines + GR.gTotalLines
```

```
  else DoneWithClusters = true
```

```
end loop when DoneWithClusters = true
```

```
dens = call feed_density
  Passing:
  areaname
  swx
  swy
  cenx
  ceny
```

swxkf
swykf

create file **DENfile** with filename {areaname}.DEN

write the value in **dens** to **DENfile**

if WeightDensity then

{ Now update density calculation for each cluster by entering a weighted average of the cluster density and the feeder density. This will handle anomalous clusters, such as a townhouse development in the middle of a farm region by weighting the small cluster's relatively high density by its population relative to the relatively low density of the overall region. }

open file **BAKfile** with filename {areaname}.BAK
create file **BINfile** with filename {areaname}.BIN

loop until reach the end of file **BAKfile**

read values from the file **BAKfile** into the variable **GR**

gtl = **GR.gTotalLines**

GR.density = (**GR.density** * **gtl** + **dens** * (TotalLines - **gtl**)) / TotalLines

write the values from the variable **GR** to the file **BINfile**

end loop

else

rename the file {areaname}.BAK to {areaname}.BIN

end if

end of procedure process

BEGIN

(start of main program)

set the following parameters from the command line:

areaname

CloseWindow

SetRaster

GlobalRasterSize

WeightDensity

if (**areaname**='BATCH') or (**areaname**='batch') then

do_batch = true

open file **batchfile** with filename 'batch.lst'

end if

if not **do_batch** then

call **process**(**areaname**)

else

for each line in **batchfile**

read line into **areaname**

call **process**(**areaname**)

next

end if

density.pas

{ This unit contains routines used to calculate density for distribution areas and feeder routes. }

function PolygonArea

passed variables:

x
y
N

local variables:

i
j
area

{This Pascal function returns the area of a polygon. It has been translated from a C routine provided by Paul Bourke on his web site,

<http://www.mhri.edu.au/~pdb/geometry/polyarea/>.

The units initially are the square of the ones used for the vertices of the polygon that is, kf-squared. We translate that into mi-squared.

Note that the vertices passed to this routine must be unique i.e., the routine will close the polygon itself. }

area = **zero**

for i = 0 to N-1

 j = (i + 1) mod N

 area = area + x[i+1] * y[j+1]

 area = area - y[i+1] * x[j+1]

next

area = area * **half**

area = abs(area)

area = area / (5.28 * 5.28) *{ convert from square kilofeet to square miles. }*

PolygonArea = area

end function PolygonArea

procedure AreaHull

passed variables:

x
y
*HullInd
NoPts
*HullCnt
*area
*HullCalc

```
local constants:
tol = 1.0e-16
```

```
local variables:
dist
t
angle
d2mn
d2im
i
j
m
n
AngleInd
hullx
hully
```

{This procedure is taken from code published on the internet by the MapTools Company.

It is an implementation of Graham's Algorithm for finding the convex hull of a set of points.

The variable HullCalc returns FALSE if NoPts < 3 or if any points are duplicate or if all points are on a line. Note that calls to this procedure must first check to see if there are any duplicate points the routine does not return useful data if duplicates exist.)

```
HullCalc = true
HullCnt = 0
```

```
if NoPts < 3 then
  HullCalc = false
else
```

```
  { Find the pair m,n with the greatest separation }
```

```
  dist = zero
  for i = 1 to NoPts
    for j = i + 1 to NoPts
      t = sqrt(x[j] - x[i]) + sqrt(y[j] - y[i])
      if t > dist then
        m = i
        n = j
        dist = t
      end if
    next j
  next i
```

```
  if abs(dist) < tol then
    HullCalc = false
  else
```

```
    { Find the rightmost point from the line m to n }
    angle = -1
    AngleInd = m
```

```

for j = 1 to 2
  d2mn = sqr(x[n] - x[m]) + sqr(y[n] - y[m])

  if abs(d2mn) < tol then
    HullCalc = false
  else
    for i = 1 to NoPts
      if ( i <> m) and (i <> n) then
        d2im = sqr(x[i] - x[m]) + sqr(y[i] - y[m])

        if abs(d2im) < tol then
          HullCalc = false
        else
          t = ((x[i]-x[m]) * (y[n]-y[m])
              -(x[n] - x[m]) * (y[i]-y[m])) / sqrt(d2im *
d2mn)

          if (t > angle ) then
            angle = t
            AngleInd = i
          end if

          if (t = angle)
            and (sqr(x[i] - x[m]) + sqr(y[i] - y[m])
                < sqr(x[AngleInd] - x[m])
                + sqr(y[AngleInd] - y[m])) then
              AngleInd = i
            end if
          end if
        end if
      end if
    next i

    { Test to make sure that some point is to the right ow,
      exchange m and n and do again }

    if angle <= zero then
      if j = 2 then
        HullCalc = false
      else
        AngleInd = n
        n = m
        m = AngleInd
      end if
    end if
  end if
next j

HullInd[1] = m
HullInd[2] = AngleInd
HullCnt = 2
n = m
m = AngleInd

```

```

    { Find the point at the greatest clockwise angle from current hull edge
}

repeat
  angle = 1
  AngleInd = m
  d2mn = sqr(x[n] - x[m]) + sqr(y[n] - y[m])

  for i = 1 to NoPts
    if (i <> m) and (i <> n) then
      d2im = sqr(x[i] - x[m]) + sqr(y[i] - y[m])
      if abs(d2im) < tol then
        HullCalc = false
      else
        t = ((x[i] - x[m]) * (x[n] - x[m])
          + (y[i] - y[m]) * (y[n] - y[m]))
          / sqrt(d2im * d2mn)

        if (t < angle) then
          angle = t
          AngleInd = i
        end if

        if (t = angle)
          and (sqr(x[i] - x[m]) + sqr(y[i] - y[m])
            < sqr(x[AngleInd] - x[m]) + sqr(y[AngleInd] -
y[m])) then
            AngleInd = i
          end if
        end if
      end if
    next i

    HullCnt = HullCnt + 1
    HullInd[HullCnt] = AngleInd
    n = m
    m = AngleInd
  until HullInd[HullCnt] = HullInd[1]
end if
end if

for i = 1 to HullCnt-1
  { Note that the last vertex is the same as the first, so we discard it for
area calc }
  HullX[i] = x[HullInd[i]]
  HullY[i] = y[HullInd[i]]
next

area = call PolygonArea
  Passing:
  HullX
  HullY
  HullCnt -1

end of procedure AreaHull

```

function MIN

passed variables:

x

y

if (x <= y) then

 min = x

else

 min = y

end if

function MAX

passed variables:

x

y

if (x >= y) then

 max = x

else

 max = y

end if

function InsidePolygon

passed variables:

x

y

N

px

py

{This function returns INSIDE (true) or OUTSIDE (false) indicating the status of a point P with respect to a polygon with N points.

The code was translated from a C routine written by Paul Bourke, posted on his website at <http://www.mhri.edu.au/~pdb/geometry/> }

local variables:

counter

i

xinters

p1x

ply

p2x

p2y

counter = 0

p1x = x[1]

ply = y[1]

for i = 0 to N-1

 p2x = x[(i mod N) + 1]

 p2y = y[(i mod N) + 1]

 if (py > **MIN**(ply , p2y)) then

 if (py <= **MAX**(ply,p2y)) then

 if (px <= **MAX**(p1x,p2x)) then

 if (ply <> p2y) then

 xinters = (py-ply)*(p2x-p1x)/(p2y-ply)+p1x

```

        if (plx = p2x) or (px <= xinters) then
            counter = counter+1
        end if
    end if
end if
end if
    end if
    plx = p2x
    ply = p2y

next

if (counter mod 2) = 0 then
    InsidePolygon = false
else
    InsidePolygon = true
end if

end function InsidePolygon

```

function dist_density

passed variables:

GR

local constant tiny = 1.0e-6

local variables:

i
j
k
x
y
HullInd
HullCnt
area
HullCalc
lines
test

{This function calculates the density for a distribution area by counting the number of lines and dividing by the area of the convex hull of the populated region in the distribution area. }

```

lines = zero
k = 1
for i = 1 to GR.ncol
    for j = 1 to GR.nrow
        if (GR.Households[i,j] + GR.BusLines[i,j]) > 0 then
            lines = lines + GR.Households[i,j] * TakeRate
                *lines_per_house + GR.BusLines[i,j]
        end if
    end for
end for

```

(We include all four corners of each microgrid in order to guarantee that all relevant area is included. We draw in the borders of each microgrid a "tiny" amount so that all points are unique, which is required by the convex hull calculation.)

```
x[k] = GR.LowerLeftX + (j-1) * GR.MicrogridEW + tiny
y[k] = GR.LowerLeftY + (i-1) * GR.MicrogridNS + tiny
```

```
x[k+1] = GR.LowerLeftX + (j-1) * GR.MicrogridEW + tiny
y[k+1] = GR.LowerLeftY + i * GR.MicrogridNS - tiny
```

```
x[k+2] = GR.LowerLeftX + j * GR.MicrogridEW - tiny
y[k+2] = GR.LowerLeftY + (i-1) * GR.MicrogridNS + tiny
```

```
x[k+3] = GR.LowerLeftX + j * GR.MicrogridEW - tiny
y[k+3] = GR.LowerLeftY + i * GR.MicrogridNS - tiny
```

```
k = k+4
```

```
end if
```

```
next j
```

```
next i
```

```
k = k-1
```

```
call AreaHull
```

```
passing:
```

```
x
```

```
y
```

```
*HullInd
```

```
k
```

```
*HullCnt
```

```
*area
```

```
*HullCalc
```

```
if HullCalc then
```

```
dist_density = lines/area
```

```
else
```

```
dist_density = zero
```

```
end if
```

```
end function dist_density
```

```
procedure reverse_convert
```

```
passed variables:
```

```
xkf
```

```
ykf
```

```
reflon
```

```
reflat
```

```
*xlon
```

```
*ylat
```

```
local constants:
```

```
EarthRadius_meters = 6367723.0
```

```
KFPerMeter = 0.00328083989501312
```

```
local variables:
```

```
NSCirc
```

```
EWCCirc
```

```
NSCirc = 2 * pi * EarthRadius_meters * KFPerMeter
```

```
EWCCirc = NSCirc*cos(reflat*pi/180.0)
```

```
ylat = ykf * 360 / NSCirc - reflat
```

```
xlon = xkf * 360 / EWCCirc - reflon
```

```
end procedure reverse_convert
```

```
function feed_density
```

```
passed variables:
```

```
areaname
```

```
swx
```

```
swy
```

```
cenx
```

```
ceny
```

```
swxkf
```

```
swykf
```

```
local variables:
```

```
BINfile
```

```
x
```

```
y
```

```
HullX
```

```
HullY
```

```
HullInd
```

```
i
```

```
j
```

```
HullCnt
```

```
area
```

```
HullCalc
```

```
lines
```

```
px
```

```
py
```

```
test
```

```
xlon
```

```
ylat
```

```
{ This function calculates the line density of the convex hull of the SAI locations. }
```

```
open the file BINfile with filename {areaname}.BAK
```

```
i = 0
```

```
loop until reach the end of file BINfile
```

```
read values from BINfile into variable GR
```

```
i = i+1
```

```
x[i] = GR.cx1[1]
```



```

        lines = lines + GR.Households[i,j] * takerate
            * Lines_Per_House + GR.BusLines[i,j]
            end if
        end if
    next j
next i
end loop

if HullCalc then
    feed_density = lines/area
else
    feed_density = zero
    Display error message: "Feeder density calculation failed"
end if

else { We have to calculate density by looking at individual grid density }

lines = zero
area = zero

for i = 1 to num_SAs do
    read values from file BINfile into GR
    area = area + GR.density * GR.gTotalLines
    lines = lines + GR.gTotalLines
next

feed_density = lines / area

End of program

```

HCPM Clustering Critique

The HCPM code was analyzed by the BCPM Sponsors to understand the methodology behind the model and how the algorithms truly work. Taking into account intricacies of writing a complex program, this point-by-point critique is meant to be constructive rather than nit picking.

The proceeding comments have been divided into categories addressing general review, programming technique, algorithm design, network design, and a review of the logic. It should be noted that the following points have been identified in a very short amount of time and will be supplemented with a more complete analysis in the future.

General Review of the HCPM Clustering Approach

The HCPM is designed to rely upon geocoded customer locations as the basis for its clustering module. Census Block data can also be input as a supplement to imperfect geocoding success rates¹. Once this data is provided, the clustering code will then generate the “engineering” areas the FeedDist program builds in and to.

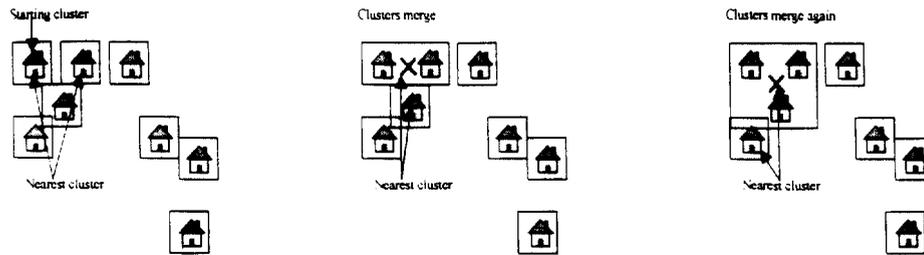
To recap briefly the clustering approach, there are two bottom-up (“agglomerative”) algorithms and one top-down (“divisive”) algorithm implemented here. Each of them honors constraints of maximum number of lines in a cluster, and maximum rectangular distance from a cluster’s centroid to any cell in the cluster².

The base bottom-up agglomerative algorithm is the standard one, with no surprises. The second, referred to as “nearest neighbor” is essentially the same as the first, with the added constraint of a two-mile maximum distance to the “nearest neighbor” of a cell in a cluster. The diagram below demonstrates either of the bottom-up agglomerative approaches.

¹ A method for geographically locating the census block data and for combining the data geocoded and surrogate data types has yet to be determined. However, as we stated in the Public Notice response, we recommend the use of roads to locate surrogate points and the use of all surrogate points whenever the success rate of the geographic area being analyzed falls below 85%.

² In some algorithms, the constraint values are user inputs; in another they are not. There does seem to be inconsistency in this.

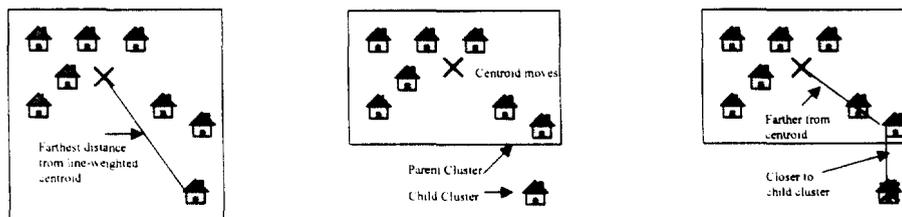
Agglomerative Approach



1. Identify the starting cluster and then determine the next nearest cluster
2. The two clusters merge. Identify the next nearest cluster to the new cluster now containing two customers.
3. The process continues until distance or total customer constraints are reached.

The divisive algorithm is also essentially a standard one, starting with all cells in a single, too-large cluster, and carving out successive clusters from it (using bottom-up procedures to agglomerate cells), always starting with the currently unassigned cell farthest away from the mother cluster's centroid. The diagram below demonstrates the divisive approach.

Divisive Approach



1. Identify the customer farthest from the line-weighted cluster centroid.
2. Farthest customer splits from the parent cluster and becomes a new child cluster.
3. Identify if the remaining customers are closer to the new child cluster centroid, or the parent centroid.

Regardless of the algorithm used, a highly iterative set of optimizing procedures follow. The sole aim of optimization is to reduce the total “noise” of the set of clusters. (For any cluster, the “noise” is the sum of the rectangular distances of each cell from the cluster centroid, each distance weighted by the number of customers at that cell location.) There are two exit criteria for terminating optimization: 1) the last iteration was unable to produce a reduction in the total noise value; or 2) a user-input time limit was exceeded³.

³ Note that on faster machines the optimization for a wire center might go to completion, while on slower machines full optimization might not complete. We can easily imagine multiple members of a commission, using identical time limits but different machines, getting different results from a model.

All in all, the user has the potential to select from 9 different methods of developing the cluster data. This implies that 9 result files can be obtained from the same input geocoded points⁴.

Critique of the Code

Programming Technique:

- Variable typing are not consistently applied. While this is probably an oversight, it does result in potential problems with the clustering algorithms. The number of rasters and the number of clusters generated are dependent on how the program variables are typed.

Type declaration is incorrect. Unlike Pascal, in VB each variable must have a type assigned to it. If it is done by line, only the last variable is of the desired type, the rest have a variant data type. For example, in the declaration:

```
Dim Lon, Lat, Area, maxArea As Double
```

Lon, Lat and Area will be declared as variants, and only maxArea will be a double.

This point is especially important in declarations like:

```
Dim Dist(), id(), Nodes, Cells As Long
```

In this case, the arrays Dist() and Id() are actually arrays of variants, not longs. This can affect memory and performance.

- It is considered poor practice to use the "+" symbol for concatenations. The "&" symbol should be used instead. When variants are used in string concatenations using "+", the result is unpredictable. If all the variables can be treated as numeric, the variables are added together, instead of concatenated.
- The code for this entire project is included in one subroutine. This makes auditing any given method difficult. The standard method of coding would dictate that each method be given its own subroutine(s). This modular approach would simplify testing and validation.
- A variable naming convention should be used. This would make the code easier to read. There are several common naming conventions that can be adopted. This point is true for the other two HCPM modules as well.

⁴ While it is nice to offer the user such a choice, it will difficult to maintain, test, verify, compare and select the appropriate results for a national model.

- The output files are used to collect processing information. This statistical information should be kept in a separate file if it is necessary. Also the cluster results and the geo-coded points could be separated to simplify downstream processing.

Technique Relating to Speed:

- Each algorithm should be reviewed. Largely, the data is processed in large square matrices when triangular structures could be used. Changing the underlying approach would probably result in improved processing time.
- File structure could be improved. The structure of the geo-coded file contains too much information. It appears that the terrain data is common for each census block. If this information were removed from the input file and stored as either a separate file or in a database, then the amount of data for each geo-coded point would be reduced substantially. The reduction in input would open alternate methods that might speed up processing.

Algorithm Design:

- Different algorithms utilize different line limits. The Divisive algorithm use lines multiplied by the line fill factor, the other two methods use straight-line counts. It is not apparent to the user that this is happening.
- The clustering methods are not consistent. In some the cases the constraints are hard-coded even though there are client inputs. The client inputs are not handled consistently. For instance, the line limits probably should be adjusted for the fill factors, but are not.
- There is no way to choose which clustering algorithm produces the 'best' results. For a given set of constraints, there should be some measure to indicate which clustering method would result in the minimum cable costs. Possibly the line weighted distance from the wire center switch location to each cluster plus the cluster line weighted distances would work.
- The graphs, while informative, should be removed from the processing. The graphs would probably work better if the clustering program had a review mode. The graphs for each method could then be reviewed at the same time.

Network Design:

- The distance limit that is not adjusted for the length of the cell. When the final raster size is determined, the distance limit should be adjusted by 1/2 of the diagonal of the raster. Without adjustment there is a potential that the distances of individual points would exceed the distance limits when the feeder/distribution plant is built.
- When a set of clusters is optimized by noise reduction the process potentially terminates due to exceeding a time constraint. One obvious drawback of this method

is that the number of iterations could vary by processor. When a process is terminated by a 'time out' condition, the results are left in the intermediate state.

Terrain:

- The terrain data for the output clusters are not correctly assigned. Terrain data is assigned to each cluster based on the terrain of the cell closest to the cluster centroid. There should be some method to develop weighted terrain data for each cluster.

Critique of Logic

Building Rasters:

The number of rasters (cells) generated is a function of the raster size and the number of populated cell constraint. A rectangle is generated that encloses the geo-coded data. This rectangle is divided into cells by dividing the rectangle side length by the raster size. The geo-coded data is then placed in a cell. If the number of populated cells is greater than the populated cell constraint, then the length of the raster is increased by the raster size and a new grid is generated.

Concerns:

1. The input file is read multiple times utilizing repeated blocks of code rather than a common subroutine.
2. Process produces different results if the data types are properly defined. In testing we found that changing the data types to consistent values generated different results. For instance, when generating rasters for the Divisive clustering algorithm the program as released generates 2754 rasters, correcting the data types results in only 2752 rasters. The number of clusters generated also changed for some of the algorithm when the data types are changed.
3. Even though the raster size is related to the distance limit and number of blocks, there is no check on user input. It appears that the number of blocks is not used in any case. As a matter of fact, the number of blocks and the distance limit are not used in the raster generating process.
4. The number of rasters is set to 3000 when the Divisive clustering algorithm is chosen.
5. The input file is read using a fixed format. The data in the input file is standard comma separated values (CSV) and should be processed as such. There is no guarantee that a particular data item will be located in a specific position in a CSV file.

Solutions:

1. Generate temporary random access file based on input file. This type of file lends itself to indexed reads. This way the data only has to be parsed once.
2. Update display when divisive algorithm is chosen.

Divisive Clustering:

Create a single cluster that contains all cells. Generate new clusters centered on cells that are furthest from the parent cluster centroid. As each new cluster is generated it collects all cells that are closer to it than the parent and still maintains the line constraint and distance constraint. Method is complete when parent cluster is either reduced to a single cell or falls within limits.

Concerns:

1. The limit on the number of rasters is set to 3000 regardless of client input. There is no notification that the change happens.

Agglomerative Clustering:

A cluster is created for each cell. Clusters are then combined in minimum distance order, subject to the line constraint and distance constraint. When the minimum distance is greater than the distance constraint the algorithm is complete.

Concerns:

1. The line constraint does not include the application of the fill factor.

Nearest Neighbor Clustering:

The algorithm is essentially the Agglomerative clustering method with the addition of a 2-mile minimum distance constraint.

Concerns:

1. The line constraint is hard coded at 1800.
2. The 2-mile limitation seems arbitrary. It is not tied to any physical limitation of telephone plant.

Optimization by Simple Reassignment:

For a given set of clusters, each cell is assigned to the nearest cluster. Besides the usual constraints there is an additional constraint. The additional constraint is the distance of the cell from the target cluster must be less than 1.5 times the distance to its original cluster. The process ends when there is no improvement in the line weighted distances.

Concerns:

1. Not sure what the additional constraint accomplishes.
2. Looks like the number of clusters can not be reduced by this process.

Optimization by Noise Reduction:

For a given set of clusters, move cells between clusters that maximize the reduction in noise gain. Noise is defined to be the number of lines times the distance to the cluster centroid.

Concerns:

1. This method is constrained by a time limit. There is no indication that the time limit is the terminating factor. The cluster is left in the modified state for subsequent processing.

Recommendation

The developers of the HCPM are in an enviable position. They have had the opportunity to assess proposed cost proxy models, namely the BCPM and HAI, in order to select the best proven approach to different issues associated with identifying the cost of supplying telephone service.

However, some inputs and approaches, which may have a great influence on the resulting outcome have yet to be determined. That being the case, we cannot make a recommendation as to whether the HCPM clustering approach is acceptable or not.

To fully understand the clustering and its impact on subsidy values, we need closure on:

- Customer Location:
 - What is the source of data
 - What is the geocoding process
 - What is the source for surrogate points
 - What is the method to determine surrogate locations
 - How should surrogate and geocoded points be combined
 - Should vacant households be included

- Clustering:
 - Should it be used
 - What approach should be used
 - What are the binding and engineering constraints
 - What is the proper raster size
 - What is the proper microgrid size

- Data for Analysis:
 - What are the results at the state, wirecenter, density group, ... levels
 - Will the code hold up as all 50 states and Puerto Rico are run through
 - How do the results compare to the existing models
 - Is the underlying data available for review

Finally, we would recommend that the code be revisited in order to make the changes that are noted in the point-by-point critique. Underlying constraints or mistakes written into the code may significantly jeopardize the accuracy of this model. In addition, as a part of the code analysis, it will be important to determine if the use of three clustering algorithms and 3 optimization routines truly adds accuracy to the results in the majority of circumstances. Or, does it add confusion to the process by allowing users to have the potential of developing 9 or more different results from the same input data.

Attachment

The HCPM code has been rewritten in Visual Basic to make further analysis more efficient. Obviously, this is a lengthy document. It will be attached separately and at a later date.