```
        SoftRockStruc[i].DistAer
```

*W – see below

```
read in the values for the array HardRockStruc from the file hardrock.txt:

for i = 1 to NumDensZones
    read in
      HardRockStruc[i].Density
      HardRockStruc[i].FeedUgd
      HardRockStruc[i].DistUgd
      HardRockStruc[i].FeedBur
      HardRockStruc[i].DistBur
      HardRockStruc[i].FeedAer
      HardRockStruc[i].DistAer
```

*W – see below

```
read in the values for the array HanholeCost from the file mhcost.txt:

NumManholeSizes = 0

for each line in the file
    NumManholeSizes = NumManholeSizes + 1
    read in
      ManholeCost[NumManholeSizes].DuctCap
      ManholeCost[NumManholeSizes].NormalCost
      ManholeCost[NumManholeSizes].SoftCost
      ManholeCost[NumManholeSizes].HardCost


read in the values for the array Manholespac from the file mhspace.txt:

for i = 1 to NumDensZones
    read in
      ManholeSpac[i].Density
      ManholeSpac[i].ManholeSpacing
```

*W – see below

```
read in the values for the array DistPlantMix from the file distrmix.txt:

for i = 1 to NumDensZones
    read in
      DistPlantMix[i].Density
      DistPlantMix[i].UgdPct
      DistPlantMix[i].BurPct
      DistPlantMix[i].AerPct
```

*W – see below

```
read in the values for the array CopFeedPlantMix from the file fdrmix.txt:
```



```
for i = 1 to NumDensZones
    read in
      CopFeedPlantMix[i].Density
      CopFeedPlantMix[i].UgdPct
      CopFeedPlantMix[i].BurPct
      CopFeedPlantMix[i].AerPct
```

*W – see below

```
read in the values for the array FibFeedPlantMix from the file fdrmix.txt:

for i = 1 to NumDensZones
    read in
      FibFeedPlantMix[i].Density
      FibFeedPlantMix[i].UgdPct
      FibFeedPlantMix[i].BurPct
      FibFeedPlantMix[i].AerPct
```

*W – The file fdrmix.txt is used to populate both the CopFeedPlantMix array, and the FibFeedPlantMix array. If these arrays are meant to be identical, then only one of the arrays should be used. If they are separate because of the possibility that they might contain different data, then separate txt files should be used to populate them. The way it is now, they will always be identical.

*W – see below

```
read in the values for the array FillFact from the file fillfact.txt:

for i = 1 to NumDensZones
    read in
      FillFact[i].Density
      FillFact[i].FeedFillFactor
      FillFact[i].DistFillFactor
```

*W – see below

```
read in the values for the array Sharing from the file sharing.txt:

for i = 1 to NumDensZones
    read in
      Sharing[i].Density
      Sharing[i].bur_share
      Sharing[i].ugd_share
      Sharing[i].aer_share
```

*W – see below

```
read in the vaules for the array SurfText from the file soiltx.txt:

NumTexTypes = 0

for each line in the file
    NumTexTypes = NumTexTypes + 1
```

```
read in
  SurfText[NumTexTypes].Texture
  SurfText[NumTexTypes].impact
```

*W – the files softrock.txt, hardrock.txt, mhspace.txt, distrmix.txt, fdrmix.txt, fillfact.txt and sharing.txt are all assumed to have the same number of density elements as the file normal.txt. This requirement is never enforced.

```
procedure get_state_data
    local variables:
    i
    infile
    }
    read the following variables from the file states.txt based on the current state:
    }
    takerate
    lines_per_house
    SpoilAccessRatio
```

```
procedure DisposeTables
```

This procedure frees up memory used by the table arrays. This is a memory management procedure only.


```
program feeddist


procedure process
    passed variables:
    areaname

    local variables:
    i
    density



    make sure the file [areaname].COO exists and has data
    open the file coordinate_file with the filename: [areaname].COO

    make sure the file [areaname].BIN exists and has data
    open the file gridfile with the filename: [areaname].BIN

    make sure the file [areaname].DEN exists and has data
    open the file DENfile with the filename: [areaname].DEN


    if both files exist and have data then continue

    read the values in coordinate_file into CR

    call get_user_parameters                                        (global.pas)
    call get_cost_data                                              (global.pas)
    call get_state_data                                             (global.pas)

    { First, calculate distribution cost, including SAI cost.  We will }
    { store all SA information in an array.                            }

    lines_served = zero
    num_SAs = 0

    for each record in the file gridfile

        num_SAs = num_SAs + 1

        read a record from the file gridfile into GR

        call optimise_SAI_arrangement                               (distrib.pas)
             pass variables:
             SwitchX = GR.SwitchX
             SwitchY = GR.SwitchY
             GR      = GR
             *SA     = SA
             *grid_lines_served = grid_lines_served
```

*W - These are all global variables being passed to and from the procedure.

```
        insert the value returned in SA into the array SA_array
        SA_array[num_SAs] = SA
```

ldist.pas

```
    lines_served = lines_served + grid_lines_served

next record

Read from DENFile into variable density


call optimize_feeder_arrangement                                    (feeder.pas)
    pass variables:
    SwitchX            = GR.SwitchX
    SwitchY            = GR.SwitchY
    density            = density
    num_SA's           = num_SAs
    *SA_array          = SA_array
    *feeder_cost       = tot_feedcost
    *feed_splice_cost  = feed_splice_cost
    *ugd_cable      = feed_ugd_cable
    *bur_cable      = feed_bur_cable
    *aer_cable      = feed_aer_cable
    *ugd_fiber      = feed_ugd_fiber
    *bur_fiber      = feed_bur_fiber
    *aer_fiber      = feed_aer_fiber
    *ugd_structrue  = feed_ugd_structure
    *bur_structure  = feed_bur_structure
    *aer_structrue  = feed_aer_structure
    *ManholeCost    = FeedManholeCost


    tot_feedcost = tot_feedcost + feed_splice_cost


  { Now collect results and print out. }
```

*W - These are all global variables being passed to and from the procedure.

```
tot_tterms          = 0
tot_fterms          = 0
Total_investment    = zero
Total_lines         = zero
tot_reslines        = zero
tot_buslines        = zero
tot_dropfeet        = zero
tot_distfeet        = zero
tot_feedfeet        = zero
tot_distcost        = zero
tot_dropcost        = zero
tot_ftcost          = zero
tot_ttcost          = zero
tot_fdicost         = zero
tot_nidcost         = zero
tot_dtcost          = zero
tot_SpclAccLines    = zero
tot_households      = zero
```


feeddist.pas

```
dist_ugd_cable      = zero
dist_bur_cable      = zero
dist_aer_cable      = zero
dist_ugd_structure  = zero
dist_bur_structure  = zero
dist_aer_structure  = zero
DistManholeCost     = zero
CalcCuFeedFill      = zero
CalcCuDistFill      = zero
tot_DLC_lines       = zero
tot_sais            = zero
tot_area            = zero


for i = 1 to num_SAs


    Total_lines = Total_lines + SA_array[i].lines

    tot_tterms  = tot_tterms  + SA_array[i].nc96 + SA_array[i].nc24
                  + SA_array[i].snc96 + SA_array[i].snc24

    tot_fterms  = tot_fterms + SA_array[i].n2016 + SA_array[i].n672
                  + SA_array[i].n96 + SA_array[i].n24

    tot_reslines = tot_reslines + SA_array[i].ResLines

    tot_buslines = tot_buslines + SA_array[i].BusLines

    tot_sais = tot_sais + SA_array[i].number_of_SAIs

    tot_area = tot_area + SA_array[i].lines / SA_array[i].density

    tot_dropfeet = tot_dropfeet + SA_array[i].grid_drop_feet

    tot_distfeet = tot_distfeet + SA_array[i].grid_line_feet

    tot_feedfeet = tot_feedfeet + SA_array[i].DistToSwitch * SA_array[i].lines

    tot_distcost = tot_distcost + SA_array[i].grid_distribution_cost

    tot_dropcost = tot_dropcost + SA_array[i].drop_cost

    tot_ftcost = tot_ftcost + SA_array[i].fiber_terminal_cost

    tot_ttcost = tot_ttcost + SA_array[i].t1_terminal_cost
                 + SA_array[i].secondary_tterm_cost

    tot_fdicost = tot_fdicost + SA_array[i].interface_cost

    tot_nidcost = tot_nidcost + SA_array[i].nid_cost

    tot_dtcost = tot_dtcost + SA_array[i].drop_terminal_cost

    tot_households = tot_households + SA_array[i].households

    tot_SpclAccLines = tot_SpclAccLines + SA_array[i].SpclAccessLines
```

```
                    + SA_array[i].SpclAccessDB1 * 12.0

    dist_bur_cable = dist_bur_cable + SA_array[i].bur_cable

    dist_aer_cable = dist_aer_cable + SA_array[i].aer_cable

    dist_ugd_cable = dist_ugd_cable + SA_array[i].ugd_cable
    dist_ugd_structure = dist_ugd_structure + SA_array[i].ugd_structure

    dist_bur_structure = dist_bur_structure + SA_array[i].bur_structure

    dist_aer_structure = dist_aer_structure + SA_array[i].aer_structure

    DistManholeCost = DistManholeCost + SA_array[i].ManholeCost

    CalcCuDistFill = CalcCuDistFill + SA_array[i].lines
                    * fill_factor_fn(SA_array[i].density,0)          (global.pas)

    if (SA_array[i].n2016 + SA_array[i].n672 + SA_array[i].n96 + SA_array[i].n24) = 0
       then CalcCuFeedFill = CalcCuFeedFill + SA_array[i].lines
                            * fill_factor_fn(SA_array[i].density,1)   (global.pas)

    if (SA_array[i].n2016 + SA_array[i].n672 + SA_array[i].n96 + SA_array[i].n24) > 0
       then tot_DLC_lines = tot_DLC_lines + SA_array[i].lines

next

Total_investment = tot_feedcost + tot_distcost + tot_dropcost + tot_fdicost
                 + tot_ttcost + tot_ftcost + tot_nidcost + tot_dtcost

average_investment = total_investment / total_lines

avg_droplength = tot_dropfeet / total_lines

avg_distlength = tot_distfeet / total_lines

avg_feedlength = tot_feedfeet / total_lines

CalcCuDistFill = CalcCuDistFill / total_lines

CalcCuFeedFill = CalcCuFeedFill / total_lines

call PrintOut

TART OF MAIN PROGRAM

heck for command line parameters - set the following variables

reaname
loseWindow
erboseOut
eaPrimDist

rimCutOffDensity = 0
```



```
if areaname = 'batch' then
    do_batch = true
    open the file batchfile with the filename 'batch.lst'
    call process for each areaname in batchfile                    (feeddist.pas)
else
    call process with areaname                                     (feeddist.pas)
end if
```

*W - Passing a global varable to the routine

```
distrib.pas

The only procedure used outside of this module is optimize_SAI_arrangement


PROCEDURE sort2vecs

    passed variables:
    n
    *ra
    *rb


    this procedure sorts the arrays ra and rb into ascending order based on the value of ra


procedure calculate_microgrid_cost

    passed variables:
    GR
    SA
    microgrid_lines
    NS_lots
    EW_lots
    gauge
    *line_vector
    *tie_in_vector
    *vdim
    *microgrid_cost
    *mg_line_feet
    *drop_feet
    *drop_cost
    *drop_terminal_cost
    *MG_nid_cost
    *ugd_cable
    *bur_cable
    *aer_cable
    *ugd_structure
    *bur_structure
    *aer_structure
    *HanholeCost


    local variables:
    i
    j
    lines
    factor
    drop_length
    lines_per_lot
    total_lots
    density
    cable_cost
    structure_cost
    fillFactor
```



```
    uc
    bc
    ac
    us
    bs
    as
    mh
    frontage
    line_feet
    pct_ugd
    pct_bur
    pct_aer


    mgcounter = mgcounter + 1

    lines            = zero
    factor           = zero
    drop_length      = zero
    lines_per_lot    = zero
    total_lots       = zero
    density          = zero
    cable_cost       = zero
    structure_cost   = zero
    mg_line_feet     = zero
    fillFactor       = zero
    uc               = zero
    bc               = zero
    ac               = zero
    us               = zero
    bs               = zero
    as               = zero
    mh               = zero
    ugd_cable        = zero
    bur_cable        = zero
    aer_cable        = zero
    ugd_structure    = zero
    bur_structure    = zero
    aer_structure    = zero
    HanholeCost      = zero

    for i = 1 to 50
      line_vector[i]   = zero
      tie_in_vector[i] = zero
    next

    density := (5.28 * 5.28) * microgrid_lines / (GR.MicroGridNS * GR.MicroGridEW)

    density := SA.density

    { Correct for fill factor }

    fillFactor = fill_factor_fn(density,0)                    (global.pas)

    if fillFactor < 1.0e-6 then stop the program. Error = 'ERROR:  fill factor too small'

    microgrid_lines =  microgrid_lines / fillFactor
```

```
total_lots = NS_lots * EW_lots
if total_lots < one  then stop the program. Error = 'ERROR:  Total lots < 1'

lines_per_lot = microgrid_lines / total_lots

{Starting at lower left of microgrid, we walk north up every other lot line,
accumulating lines and cable.  If we accumulate enough lines for a
new cable, we add it, repeating the exercise until we reach either
the midpoint of the northern boundary.}

microgrid_cost = zero
drop_terminal_cost = zero
MG_nid_cost = zero
drop_cost = zero
lines = zero
line_feet = zero
drop_feet = zero
i = 1
vdim = 0

loop while i <= EW_lots
    factor = one
    lines = zero
    j = 1

    loop while j <= NS_lots
    { take in lots on both sides, top and bottom, unless this is a microgrid }
    { border, in which case take in lots only on one side.  If it is the     }
    { corner, take in only one lot.                                          }

        //If at the top of grid OR the far right of grid, set factor to 2 lots
        if i = EW_lots or j = NS_lots then
            factor = 2
        else
            factor = 4
        end if

        //If at the top AND the far right of the grid (corner), set factor to 1 lot
        if i = EW_lots and j = NS_lots then factor = one

        lines = lines + factor * lines_per_lot

        structure_cost = call structure_cost_fn                (structur.pas)
                         pass variables:
                         copper_lines      = lines
                         fiber_lines       = 0
                         density           = density
                         hardness          = GR.hardness
                         depth_to_bedrock  = GR.DepthToBedrock
                         soil_texture      = GR.SoilTexture
                         MinSlope          = GR.MinSlope
                         MaxSlope          = GR.MaxSlope
                         WaterTb           = GR.WaterTb
                         feeder_indicator  = 0
                         cooper_indicator  = 1
                         fiber_indicator   = 0
```



```
                         *ugd_structure    = us
                         *bur_structure    = bs
                         *aer_structure    = as
                         *manhole_cost     = mh
                         *pct_ugd          = pct_ugd
                         *pct_bur          = pct_bur
                         *pct_aer          = pct_aer

        cable_cost = call dist_cable_cost                          (cable.pas)
                     pass variables:
                     lines        = lines
                     denisty      = density
                     gauge        = gauge
                     *ugd_copper  = uc
                     *bur_copper  = bc
                     *aer_copper  = ac
                     pct_ugd      = pct_ugd
                     pct_bur      = pct_bur
                     pct_aer      = pct_aer

        if j <= (NS_lots - 2) then                          {frontage of 2 lots}

            frontage = (2 / NS_lots) * GR.MicroGridNS * DistRoadFactor

            microgrid_cost = microgrid_cost + frontage
                             * (cable_cost + structure_cost)
            ugd_cable = ugd_cable + uc * frontage
            bur_cable = bur_cable + bc * frontage
            aer_cable = aer_cable + ac * frontage
            ugd_structure = ugd_structure + us * frontage
            bur_structure = bur_structure + bs * frontage
            aer_structure = aer_structure + as * frontage
            ManholeCost   = ManholeCost + mh * frontage

            line_feet = line_feet + frontage * lines
            drop_terminal_cost = drop_terminal_cost + call drop_terminal_cost_fn
                                                          (terminal.pas)
                                                 pass variables:
                                                 lines   = factor * lines_per_lot
                                                 density = density
                                                 pct_ugd = pct_ugd
                                                 pct_bur = pct_bur
                                                 pct_aer = pct_aer

        else if j = (NS_lots - 1) then     {frontage of 1 lot}

            frontage = (1 / NS_lots) * GR.MicroGridNS * DistRoadFactor

            microgrid_cost = microgrid_cost + frontage
                             * (cable_cost + structure_cost)

            ugd_cable = ugd_cable + uc * frontage
            bur_cable = bur_cable + bc * frontage
            aer_cable = aer_cable + ac * frontage
            ugd_structure = ugd_structure + us * frontage
            bur_structure = bur_structure + bs * frontage
            aer_structure = aer_structure + as * frontage
```

```
        HanholeCost = HanholeCost + mh * frontage

        line_feet = line_feet + frontage * lines
        drop_terminal_cost = drop_terminal_cost + call drop_terminal_cost_fn
                                                    (terminal.pas)
                                                pass variables:
                                                lines   = factor * lines_per_lot
                                                density = density
                                                pct_ugd = pct_ugd
                                                pct_bur = pct_bur
                                                pct_aer = pct_aer


      else if (j = NS_lots) then  (at the border, only drop terminals; no cabling)

        drop_terminal_cost = drop_terminal_cost + call drop_terminal_cost_fn
                                                    (terminal.pas)
                                                pass variables:
                                                lines =  factor * lines_per_lot
                                                density = density
                                                pct_ugd = pct_ugd
                                                pct_bur = pct_bur
                                                pct_aer = pct_aer

      //move up 2 lot lines
      j = j + 2
    end of j loop  (loop while j <= NS_lots)

    mg_line_feet = mg_line_feet + line_feet
    line_feet = zero
    vdim = vdim + 1
    line_vector[vdim] = lines
    tie_in_vector[vdim] = (1 / EW_lots) * i * GR.MicrogridEW

    //move over two lot lines
    i = i + 2
end of i loop  (loop while i <= EW_lots)


{ Now we need to calculate drops to customer locations }

drop_length = user_lambda * 0.5
              * sqrt( sqr((1 / NS_lots) * GR.MicroGridNS * DistRoadFactor )
                    + sqr((1 / EW_lots) * GR.MicroGridEW * DistRoadFactor ))
              + (1 - user_lambda) * 0.5 * (1 / NS_lots) * GR.MicroGridNS
              * DistRoadFactor


if drop_length > max_drop_length then drop_length = max_drop_length

drop_cost = total_lots * drop_length * cost_per_drop_kf
drop_feet = total_lots * drop_length

{ Finally, calculate cost of nids for this microgrid }

MG_nid_cost = nid_cost * total_lots
```



```
procedure calculate_grid_distribution_cost

    passed variables: '
    GR
    SA
    number_of_SAIs
    SAIX
    SAIY
    *SAI_lines
    *grid_distribution_cost
    *grid_line_feet
    *link_line_feet
    *grid_drop_feet
    *density
    *grid_drop_cost
    *grid_terminal_cost
    *grid_nid_cost
    *dist_lines_served
    *link_cost
    *term_cost
    *nc96
    *nc24
    *MaximumDistance
    *ugd_cable
    *bur_cable
    *aer_cable
    *ugd_structure
    *bur_structure
    *aer_structure
    *HanholeCost


    local variables:
    i
    j
    n
    k
    microgrid_cost
    backbone_lines
    main_back_lines
    lines
    flag
    midx
    midy
    mindist
    divider_col
    divider_row
    lots
    microgrid_lines
    EW_lots
    NS_lots
    microgrid_line_feet
    microgrid_drop_feet
```

```
area
cable_cost
structure_cost
microgrid_drop_cost
microgrid_nid_cost
microgrid_terminal_cost
total_lines
rows_completed
rc
tl_lines
FillFactor
line_vector
line_vector1
line_vector2
tie_in_vector
tie_in_vector1
tie_in_vector2
vdim1
vdim2
MaxDist
n96
n24
gauge
penalty
uc
bc
ac
us
bs
as
mh
prim_distribution_cost
prim_ugd_cable
prim_bur_cable
prim_aer_cable
prim_ugd_structure
prim_bur_structure
prim_aer_structure
prim_ManholeCost
prim_line_feet
prim_drop_feet
prim_drop_cost
prim_nid_cost
prim_lines_served
prim_term_cost
prim_MaximumDistance
test
pct_ugd
pct_bur
pct_aer
```

This is a procedure within the procedure

```
procedure accumulate_backbone
```



```
local variables
k
pct_ugd
pct_bur
pct_aer


vdim1 = 0
vdim2 = 0

if (i > 0) and (flag[i,j] = n) and (lines[i,j] > 0)
    and (i > rows_completed) then

  { if microgrid below is populated }

    lots = round(GR.households[i,j] * takerate)
         + round(GR.buslines[i,j] / lines_per_bus)

    microgrid_lines = lines[i,j]

    call lot_divide(lots, NS_lots, EW_lots )                    (lotdiv.pas)

    call calculate_microgrid_cost                               (distrib.pas)
    pass variables:
    GR                    = GR
    SA                    = SA
    microgrid_lines       = microgrid_lines
    NS_lots               = NS_lots
    EW_lots               = EW_lots
    gauge                 = gauge
    *line_vector          = line_vector1
    *tie_in_vector        = tie_in_vector1
    *vdim                 = vdim1
    *microgrid_cost       = microgrid_cost
    *mg_line_feet         = microgrid_line_feet
    *drop_feet            = microgrid_drop_feet
    *drop_cost            = microgrid_drop_cost
    *drop_terminal_cost   = microgrid_terminal_cost
    *MG_nid_cost          = microgrid_nid_cost
    *ugd_cable            = uc
    *bur_cable            = bc
    *aer_cable            = ac
    *ugd_structure        = us
    *bur_structure        = bs
    *aer_structure        = as
    *ManholeCost          = mh

    grid_distribution_cost = grid_distribution_cost + microgrid_cost * penalty
```

*W - need to see if penalty is being applied twice

```
    grid_drop_cost = grid_drop_cost + microgrid_drop_cost

    grid_terminal_cost = grid_terminal_cost + microgrid_terminal_cost

    grid_nid_cost = grid_nid_cost + microgrid_nid_cost
```

```
    grid_line_feet = grid_line_feet + microgrid_line_feet

    grid_drop_feet = grid_drop_feet + microgrid_drop_feet

    ugd_cable = ugd_cable + uc * penalty
    bur_cable = bur_cable + bc * penalty
    aer_cable = aer_cable + ac * penalty
    ugd_structure = ugd_structure + us * penalty
    bur_structure = bur_structure + bs * penalty
    aer_structure = aer_structure + as * penalty
    ManholeCost = ManholeCost + mh * penalty
    uc = zero
    bc = zero
    ac = zero
    us = zero
    bs = zero
    as = zero
    mh = zero

end if


if ((i + 1) <= GR.nrow) and (flag[i+1,j] = n)
  and (lines[i+1,j] > 0) and ((i+1) > rows_completed ) then

  { if microgrid above is populated }

    lots = round(GR.households[i+1,j] * takerate)
         + round(GR.buslines[i+1,j] / lines_per_bus)

    microgrid_lines = lines[i+1,j]

    call lot_divide(lots, NS_lots, EW_lots )                    (lotdiv.pas)

    call calculate_microgrid_cost                               (distrib.pas)
    pass variables:
    GR                    - GR
    SA                    - SA
    microgrid_lines       - microgrid_lines
    NS_lots               - NS_lots
    EW_lots               - EW_lots
    gauge                 - gauge
    *line_vector          - line_vector2
    *tie_in_vector        - tie_in_vector2
    *vdim                 - vdim2
    *microgrid_cost       - microgrid_cost
    *mg_line_feet         - microgrid_line_feet
    *drop_feet            - microgrid_drop_feet
    *drop_cost            - microgrid_drop_cost
    *drop_terminal_cost   - microgrid_terminal_cost
    *MG_nid_cost          - microgrid_nid_cost
    *ugd_cable            - uc
    *bur_cable            - bc
    *aer_cable            - ac
    *ugd_structure        - us
    *bur_structure        - bs
    *aer_structure        - as
```



```
    *ManholeCost          - mh

    grid_distribution_cost = grid_distribution_cost + microgrid_cost * penalty

    grid_drop_cost = grid_drop_cost + microgrid_drop_cost

    grid_terminal_cost = grid_terminal_cost + microgrid_terminal_cost

    grid_nid_cost = grid_nid_cost + microgrid_nid_cost

    grid_line_feet = grid_line_feet + microgrid_line_feet

    grid_drop_feet = grid_drop_feet + microgrid_drop_feet

    ugd_cable = ugd_cable + uc * penalty
    bur_cable = bur_cable + bc * penalty
    aer_cable = aer_cable + ac * penalty
    ugd_structure = ugd_structure + us * penalty
    bur_structure = bur_structure + bs * penalty
    aer_structure = aer_structure + as * penalty
    ManholeCost = ManholeCost + mh * penalty
    uc = zero
    bc = zero
    ac = zero
    us = zero
    bs = zero
    as = zero
    mh = zero

end if

if vdim1 > 0 then

    for k = 1 to vdim1
        line_vector[k] = line_vector1[k]
        tie_in_vector[k] = tie_in_vector1[k]
        dist_lines_served = dist_lines_served + line_vector1[k]
    next

if vdim2 > 0 then

    for k = 1 to vdim2
        line_vector[vdim1+k] = line_vector2[k]
        tie_in_vector[vdim1+k] = tie_in_vector2[k]
        dist_lines_served = dist_lines_served + line_vector2[k]
    next

if (vdim1+vdim2) > 1 then

    call sort2vecs(vdim1+vdim2 ,tie_in_vector, line_vector)      (distrib.pas)

end if

{Bring forward lines from previous microgrids }

structure_cost = call structure_cost_fn                          (structur.pas)
                 pass variables:
```

```
                    backbone_lines
                    0
                    density
                    GR.hardness
                    GR.DepthToBedrock
                    GR.SoilTexture
                    GR.MinSlope
                    GR.MaxSlope
                    GR.WaterTb
                    0
                    1
                    0
                    *us
                    *bs
                    *as
                    *mh
                    *pct_ugd
                    *pct_bur
                    *pct_aer

cable_cost = call dist_cable_cost                    {cable.pas}
             pass variables:
             backbone_lines
             density
             gauge
             *uc
             *bc
             *ac
             pct_ugd
             pct_bur
             pct_aer

if (vdim1 + vdim2) > 0 then
    grid_distribution_cost = grid_distribution_cost
                           + abs(tie_in_vector[1]) * DistRoadFactor
                           *(cable_cost + structure_cost) * penalty

    ugd_cable = ugd_cable + uc * abs(tie_in_vector[1]) * DistRoadFactor
              * penalty

    bur_cable = bur_cable + bc * abs(tie_in_vector[1]) * DistRoadFactor
              * penalty

    aer_cable = aer_cable + ac * abs(tie_in_vector[1]) * DistRoadFactor
              * penalty

    ugd_structure = ugd_structure + us * abs(tie_in_vector[1]) * DistRoadFactor
                  * penalty

    bur_structure = bur_structure + bs * abs(tie_in_vector[1]) * DistRoadFactor
                  * penalty

    aer_structure = aer_structure + as * abs(tie_in_vector[1]) * DistRoadFactor
                  * penalty
```



```
    HanholeCost = HanholeCost + mh * abs(tie_in_vector[1]) * DistRoadFactor
                * penalty

    uc = zero
    bc = zero
    ac = zero
    us = zero
    bs = zero
    as = zero
    mh = zero

    grid_line_feet = grid_line_feet + backbone_lines
                   * abs(tie_in_vector[1]) * DistRoadFactor

else

   { No lines here, so cross microgrid }

    grid_distribution_cost = grid_distribution_cost + abs(GR.MicroGridEW)
                           * DistRoadFactor * (cable_cost + structure_cost)
                           * penalty

    ugd_cable = ugd_cable + uc * abs(GR.MicroGridEW) * DistRoadFactor * penalty

    bur_cable = bur_cable + bc * abs(GR.MicroGridEW) * DistRoadFactor * penalty

    aer_cable = aer_cable + ac * abs(GR.MicroGridEW) * DistRoadFactor * penalty

    ugd_structure = ugd_structure + us * abs(GR.MicroGridEW) * DistRoadFactor
                  * penalty

    bur_structure = bur_structure + bs * abs(GR.MicroGridEW) * DistRoadFactor
                  * penalty

    aer_structure = aer_structure + as * abs(GR.MicroGridEW) * DistRoadFactor
                  * penalty

    HanholeCost = HanholeCost + mh * abs(GR.MicroGridEW) * DistRoadFactor
                * penalty

    uc = zero
    bc = zero
    ac = zero
    us = zero
    bs = zero
    as = zero
    mh = zero

    grid_line_feet = grid_line_feet + backbone_lines * abs(GR.MicroGridEW)
                   * DistRoadFactor
end if

{ Capture lines from these microgrids }

if (vdim1 + vdim2) > 0 then
```

```
backbone_lines = backbone_lines + line_vector[i]

if (vdim1 + vdim2) >= 2 then

    for k = 2 to vdim1 + vdim2
        structure_cost = call structure_cost_fn          (structur.pas)
                         pass variables:
                         backbone_lines
                         0
                         density
                         GR.hardness
                         GR.DepthToBedrock
                         GR.SoilTexture
                         GR.MinSlope
                         GR.MaxSlope
                         GR.WaterTb
                         0
                         1
                         0
                         *us
                         *bs
                         *as
                         *mh
                         *pct_ugd
                         *pct_bur
                         *pct_aer


        cable_cost = call dist_cable_cost                (cable.pas)
                     pass variables:
                     backbone_lines
                     density
                     gauge
                     *uc
                     *bc
                     *ac
                     pct_ugd
                     pct_bur
                     pct_aer

        grid_distribution_cost = grid_distribution_cost +
                                 abs(tie_in_vector[k] - tie_in_vector[k-1])
                                 * DistRoadFactor
                                 * (cable_cost + structure_cost)* penalty

        ugd_cable = ugd_cable + uc
                  * abs(tie_in_vector[k] - tie_in_vector[k-1])
                  * DistRoadFactor * penalty

        bur_cable = bur_cable + bc
                  * abs(tie_in_vector[k] - tie_in_vector[k-1])
                  * DistRoadFactor * penalty

        aer_cable = aer_cable + ac
                  * abs(tie_in_vector[k] - tie_in_vector[k-1])
                  * DistRoadFactor * penalty
```



```
        ugd_structure = ugd_structure + us
                      * abs(tie_in_vector[k] - tie_in_vector[k-1])
                      * DistRoadFactor * penalty

        bur_structure = bur_structure + bs
                      * abs(tie_in_vector[k] - tie_in_vector[k-1])
                      * DistRoadFactor * penalty

        aer_structure = aer_structure + as
                      * abs(tie_in_vector[k] - tie_in_vector [k-1])
                      * DistRoadFactor * penalty

        ManholeCost = ManholeCost + mh
                    * abs(tie_in_vector[k] - tie_in_vector[k-1])
                    * DistRoadFactor * penalty

        uc = zero
        bc = zero
        ac = zero
        us = zero
        bs = zero
        as = zero
        mh = zero

        grid_line_feet = grid_line_feet + backbone_lines
                       * abs(tie_in_vector[k] - tie_in_vector[k-1])
                       * DistRoadFactor

        backbone_lines = backbone_lines + line_vector[k]

    next k

end if ( if(vdim1 + vdim2) >= 2)

{ Bring forward lines to next microgrids }

structure_cost = call structure_cost_fn                  (structur.pas)
                 pass variables:
                 backbone_lines
                 0
                 density
                 GR.hardness
                 GR.DepthToBedrock
                 GR.SoilTexture
                 GR.MinSlope
                 GR.MaxSlope
                 GR.WaterTb
                 0
                 1
                 0
                 *us
                 *bs
                 *as
                 *mh
                 *pct_ugd
                 *pct_bur
                 *pct_aer
```

```
cable_cost = call dist_cable_cost                    (cable.pas)
             pass variables:
             backbone_lines
             density
             gauge
             *uc
             *bc
             *ac
             pct_ugd
             pct_bur
             pct_aer

grid_distribution_cost = grid_distribution_cost
                       + abs(GR.MicroGridEW - tie_in_vector[vdim1+vdim2])
                       * DistRoadFactor * (cable_cost + structure_cost)
                       * penalty


ugd_cable = ugd_cable + uc
          * abs(GR.MicroGridEW - tie_in_vector[vdim1+vdim2])
          * DistRoadFactor * penalty

bur_cable = bur_cable + bc
          * abs(GR.MicroGridEW - tie_in_vector[vdim1+vdim2])
          * DistRoadFactor * penalty

aer_cable = aer_cable + ac
          * abs(GR.MicroGridEW - tie_in_vector^[vdim1+vdim2])
          * DistRoadFactor * penalty

ugd_structure = ugd_structure + us
              * abs(GR.MicroGridEW - tie_in_vector[vdim1+vdim2])
              * DistRoadFactor * penalty

bur_structure = bur_structure + bs
              * abs(GR.MicroGridEW - tie_in_vector[vdim1+vdim2])
              * DistRoadFactor * penalty

aer_structure = aer_structure + as
              * abs(GR.MicroGridEW - tie_in_vector[vdim1+vdim2])
              * DistRoadFactor * penalty

ManholeCost = ManholeCost + mh
            * abs(GR.MicroGridEW - tie_in_vector[vdim1+vdim2])
            * DistRoadFactor * penalty

uc = zero
bc = zero
ac = zero
us = zero
bs = zero
as = zero
mh = zero
```



```
            grid_line_feet = grid_line_feet + backbone_lines
                           * abs(GR.MicroGridEW - tie_in_vector^[vdim1+vdim2])
                           * DistRoadFactor

            vdim1 = 0
            vdim2 = 0

        end if (if (vdim1+vdim2) > 0)


end procedure calculate_grid_distribution_cost


(main procedure )

mgcounter                 = 0
microgrid_cost            = zero
backbone_lines            = zero
main_back_lines           = zero
midx                      = zero
midy                      = zero
mindist                   = zero
lots                      = zero
microgrid_lines           = zero
EW_lots                   = zero
NS_lots                   = zero
microgrid_line_feet       = zero
microgrid_drop_feet       = zero
area                      = zero
cable_cost                = zero
structure_cost            = zero
microgrid_drop_cost       = zero
microgrid_nid_cost        = zero
microgrid_terminal_cost   = zero
total_lines               = zero
grid_distribution_cost    = zero
grid_line_feet            = zero
link_line_feet            = zero
grid_drop_feet            = zero
density                   = zero
grid_drop_cost            = zero
grid_terminal_cost        = zero
grid_nid_cost             = zero
dist_lines_served         = zero

divider_col               = 0
divider_row               = 0
rows_completed            = 0
rc                        = 0
nc96                      = 0
nc24                      = 0

uc                        = zero
bc                        = zero
ac                        = zero
us                        = zero
bs                        = zero
```

```
    as                          = zero
    mh                          = zero
    ugd_cable                   = zero
    bur_cable                   = zero
    aer_cable                   = zero
    ugd_structure               = zero
    bur_structure               = zero
    aer_structure               = zero
    ManholeCost                 = zero
    link_line_feet              = zero

    for i = 1 to 50
        line_vector[i] = zero
        line_vector1[i] = zero
        line_vector2[i] = zero
        tie_in_vector[i] = zero
        tie_in_vector1[i] = zero
        tie_in_vector2[i] = zero
    next

    for i = 1 to 50
        for j = 1 to 50
            lines[i,j] = zero
            flag[i,j] = 0
        next
    next

    area = zero
    total_lines = zero
    density = zero

    //cumulate lines and area for the grid
    for i = 1 to GR.nrow
        for j = 1 to GR.ncol
            lines[i,j] = GR.households[i,j] * takerate * lines_per_house
                       + GR.buslines[i,j]
                        //looks like this is an attempt to remove DSl lines
                       - (11 / 12) * (1 - SpoiAccessRatio)* pot_dsl * GR.buslines[i,j]
                       - (11 / 12) * pot_lsa * SpoiAccessRatio * GR^.buslines[i,j]
```

W - GR.buslines was modified in the optimize_SAI_arrangement procedure

```
            if lines[i,j] > 0 then
                area = area + GR.MicroGridEW * GR.MicroGridNS / (5.28 * 5.28)

            total_lines = total_lines + lines[i,j]
        next j
    next i

    if area > 0 then
        density = total_lines / area
    else
        density = zero
    end if

    {First, we need to determine which microgrids are attached to which SAI }
    for i = 1 to GR.nrow
```



```
            for j = 1 to GR.ncol
                //flag is used to indicate which SAI
                flag[i,j] = 1
                midx = GR.LowerLeftX + (j - 0.5) * GR.MicroGridEW
                midy = GR.LowerLeftY + (i - 0.5) * GR.MicroGridNS
                mindist = abs(midx - SAIX[1]) + abs(midy - SAIY[1])

                for n = 2 to number_of_SAIs
                    if (abs(midx - SAIX[n]) + abs(midy - SAIY[n])) < mindist then
                        mindist = abs(midx - SAIX[n]) + abs(midy - SAIY[n])
                        flag[i,j] = n
                    end if
                next n
            next j
        next i

        for n = 1 to number_of_SAIs

            SAI_lines[n] = zero
            MaxDist[n] = zero

            for i = 1 to GR.nrow
                for j = 1 to GR.ncol

                    if (flag[i,j] = n ) and (lines[i,j] > 0) then
                        SAI_lines[n] = SAI_lines[n] + lines[i,j]
                        //why don't these calculations use the same format as above?
                        midx = GR.LowerLeftX + j * GR.MicroGridEW - half * GR.MicroGridEW
                        midy = GR.LowerLeftY + i * GR.MicroGridNS - half * GR.MicroGridNS
                        //should use global max() function
                        if (abs(midx - SAIX[n]) + abs(midy - SAIY[n]))
                            * DistRoadFactor > MaxDist^[n] then
                                MaxDist[n] = (abs(midx - SAIX[n]) + abs(midy - SAIY[n]))
                                             * DistRoadFactor
                        end if

                    end if
                next j
            next i
        next n
```

*W - Not necessary to loop throught entire grid structure twice to find min, max distances

```
        MaximumDistance = MaxDist[1]

        for n = 1 to number_of_SAIs

            if MaxDist[n] > MaximumDistance then MaximumDistance = MaxDist[n]

        next n

        { Now flag points to the nearest SAI for each microgrid cell.          }
        { We are now ready to calculate backbones.                             }

        grid_drop_cost = zero
        grid_terminal_cost = zero
        grid_nid_cost = zero
```

```
grid_distribution_cost = zero
grid_line_feet = zero
grid_drop_feet = zero
dist_lines_served = zero

for n = 1 to number_of_SAIs

    { First, calculate the gauge needed. }
    if MaxDist[n] > copper_gauge_xover then
        gauge = g24
    else
        gauge = g26
    end if

    { Now we handle any situations where customers are too far from an SAI }

    if MaxDist[n] > max_copper_distance then
        penalty = MaxCopperPenalty
    else
        penalty = one
    end if


    { We now need to divide the SA into quadrants with the SAI      }
    { serving as the "origin."                                      }

    divider_col = round(abs(SAIX[n] - GR.LowerLeftX) / GR.MicroGridEW)
    divider_row = round(abs(SAIY[n] - GR.LowerLeftY) / GR.MicroGridNS)


    { Below, we calculate for the region south of the SAI }

    i = 1
    main_back_lines = zero
    vdim1 = 0
    vdim2 = 0
    rows_completed = 0
    rc = 0

    loop while i <= divider_row

        rc = i + 1
        backbone_lines = zero
        vdim1 = 0
        vdim2 = 0

        if divider_col > 0 then
            for j = 1 to divider_col     {from western border to SAI column}
                call accumulate_backbone                                {distrib.pas}
            next
        end if

        main_back_lines = main_back_lines + backbone_lines
        backbone_lines = zero
        vdim1 = 0
        vdim2 = 0
```



```
        if GR.ncol > divider_col then
            for j = GR.ncol downto divider_col + 1 {from eastern border to SAI column}
                accumulate_backbone                                     {distrib.pas}
            next
        end if


        main_back_lines = main_back_lines + backbone_lines

        if i = divider_row - 1 then  {only one microgrid depth}
            structure_cost = call structure_cost_fn                     {structur.pas}
                             pass variables:
                             main_back_lines
                             0
                             density
                             GR.hardness
                             GR.DepthToBedrock
                             GR.SoilTexture
                             GR.MinSlope
                             GR.MaxSlope
                             GR.WaterTb
                             0
                             1
                             0
                             *us
                             *bs
                             *as
                             *mh
                             *pct_ugd
                             *pct_bur
                             *pct_aer

                cable_cost = call dist_cable_cost                       {cable.pas}
                             pass variables:
                             main_back_lines
                             density
                             gauge
                             *uc
                             *bc
                             *ac
                             pct_ugd
                             pct_bur
                             pct_aer

                grid_distribution_cost = grid_distribution_cost + GR.MicroGridNS
                                       * DistRoadFactor
                                       * (cable_cost + structure_cost) * penalty
```

*W - looks like cable distance penalty is also applied to structure costs, also why isn't penalty applied to cable over critical distance

```
                grid_line_feet = grid_line_feet + main_back_lines
                               * GR.MicroGridNS * DistRoadFactor

                ugd_cable = ugd_cable + uc * GR.MicroGridNS * DistRoadFactor
                          * penalty
```

```
        bur_cable = bur_cable + bc * GR.MicroGridNS * DistRoadFactor
                  * penalty
        aer_cable = aer_cable + ac * GR.MicroGridNS * DistRoadFactor
                  * penalty

        ugd_structure = ugd_structure + us * GR.MicroGridNS
                      * DistRoadFactor * penalty

        bur_structure = bur_structure + bs * GR.MicroGridNS
                      * DistRoadFactor * penalty

        aer_structure = aer_structure + as * GR.MicroGridNS
                      * DistRoadFactor * penalty

        ManholeCost = ManholeCost + mh * GR.MicroGridNS * DistRoadFactor
                    * penalty

else if i <> divider_row then          (two microgrid depth)

        structure_cost = call structure_cost_fn                (structur.pas)
                         pass variables:
                         main_back_lines
                         0
                         density
                         GR.hardness
                         GR.DepthToBedrock
                         GR.SoilTexture
                         GR.MinSlope
                         GR.MaxSlope
                         GR.WaterTb
                         0
                         1
                         0
                         *us
                         *bs
                         *as
                         *mh
                         *pct_ugd
                         *pct_bur
                         *pct_aer

        cable_cost = call dist_cable_cost                    (cable.pas)
                     pass variables:
                     main_back_lines
                     density
                     gauge
                     *uc
                     *bc
                     *ac
                     pct_ugd
                     pct_bur
                     pct_aer

        grid_distribution_cost = grid_distribution_cost
                               + GR.MicroGridNS * DistRoadFactor * 2
                               * (cable_cost +  structure_cost) * penalty
```



```
        grid_line_feet = grid_line_feet + 2 * main_back_lines
                       * GR.MicroGridNS * DistRoadFactor

        ugd_cable = ugd_cable + uc * 2 * GR.MicroGridNS * DistRoadFactor
                  * penalty

        bur_cable = bur_cable + bc * 2 * GR.MicroGridNS DistRoadFactor
                  * penalty

        aer_cable = aer_cable + ac * 2 * GR.MicroGridNS DistRoadFactor
                  * penalty

        ugd_structure = ugd_structure + us * 2 * GR.MicroGridNS
                      * DistRoadFactor * penalty

        bur_structure = bur_structure + bs * 2 * GR.MicroGridNS
                      * DistRoadFactor * penalty

        aer_structure = aer_structure + as * 2 * GR.MicroGridNS
                      * DistRoadFactor * penalty

        ManholeCost = ManholeCost + mh * 2 * GR.MicroGridNS
                    * DistRoadFactor * penalty
    end if

    i = i + 2    ( run backbone down every other row )

end loop (loop while i <= divider_row)


rows_completed = rc

( Now we need to calculate for the region north of the SAI )

i = GR.nrow - 1
main_back_lines = zero
vdim1 = 0
vdim2 = 0

loop while (i >= rows_completed)
    backbone_lines = zero
    vdim1 = 0
    vdim2 = 0

    if divider_col > 0 then
        for j = 1 to divider_col    ( from western border to SAI column )
            call accumulate_backbone                          (distrib.pas)
        next
    end if


    main_back_lines = main_back_lines + backbone_lines
    backbone_lines = zero
    vdim1 = 0
    vdim2 = 0
```

```
if GR.ncol > divider_col then
    for j = GR.ncol downto divider_col + 1 {from eastern border to SAI column}

        call accumulate_backbone                    {distrib.pas}
    next
end if

main_back_lines = main_back_lines + backbone_lines

if (i = divider_row + 1) then          {only one microgrid depth }
    structure_cost = call structure_cost_fn          {structur.pas}
                     pass variables:
                     main_back_lines
                     0
                     density
                     GR.hardness
                     GR.DepthToBedrock
                     GR.SoilTexture
                     GR.MinSlope
                     GR.MaxSlope
                     GR.WaterTb
                     0
                     1
                     0
                     *us
                     *bs
                     *as
                     *mh
                     *pct_ugd
                     *pct_bur
                     *pct_aer

    cable_cost = call dist_cable_cost                {cable.pas}
                 pass variables:
                 main_back_lines
                 density
                 gauge
                 *uc
                 *bc
                 *ac
                 pct_ugd
                 pct_bur
                 pct_aer

    grid_distribution_cost = grid_distribution_cost + GR.MicroGridNS
                           * DistRoadFactor
                           * (cable_cost + structure_cost) * penalty

    grid_line_feet = grid_line_feet + main_back_lines
                   * GR.MicroGridNS * DistRoadFactor

    ugd_cable = ugd_cable + uc * GR.MicroGridNS * DistRoadFactor
              * penalty

    bur_cable = bur_cable + bc * GR.MicroGridNS * DistRoadFactor
              * penalty
```



```
    aer_cable = aer_cable + ac * GR.MicroGridNS * DistRoadFactor
              * penalty
    ugd_structure = ugd_structure + us * GR.MicroGridNS * DistRoadFactor
                  * penalty

    bur_structure = bur_structure + bs * GR.MicroGridNS * DistRoadFactor
                  * penalty

    aer_structure = aer_structure + as * GR.MicroGridNS * DistRoadFactor
                  * penalty

    HanholeCost = HanholeCost + mh * GRMicroGridNS * DistRoadFactor
                * penalty


else if i <> divider_row then      {two microgrid depth}

   structure_cost = call structure_cost_fn              {structur.pas}
                    pass variables:
                    main_back_lines
                    0
                    density
                    GR.hardness
                    GR.DepthToBedrock
                    GR.SoilTexture
                    GR.MinSlope
                    GR.MaxSlope
                    GR.WaterTb
                    0
                    1
                    0
                    *us
                    *bs
                    *as
                    *mh
                    *pct_ugd
                    *pct_bur
                    *pct_aer

    cable_cost = call dist_cable_cost                   {cable.pas}
                 pass variables:
                 main_back_lines
                 density
                 gauge
                 *uc
                 *bc
                 *ac
                 pct_ugd
                 pct_bur
                 pct_aer

    grid_distribution_cost = grid_distribution_cost + GR.MicroGridNS
                           * 2 * DistRoadFactor
                           * (cable_cost + structure_cost) * penalty

    grid_line_feet = grid_line_feet + 2 * main_back_lines * GR.MicroGridNS
                   * DistRoadFactor
```

```
            ugd_cable = ugd_cable + uc * 2 * GR.MicroGridNS * DistRoadFactor
                      * penalty

            bur_cable = bur_cable + bc * 2 * GR.MicroGridNS * DistRoadFactor
                      * penalty

            aer_cable = aer_cable + ac * 2 * GR.MicroGridNS * DistRoadFactor
                      * penalty

            ugd_structure = ugd_structure + us * 2 * GR.MicroGridNS * DistRoadFactor
                          * penalty

            bur_structure = bur_structure + bs * 2 * GR.MicroGridNS * DistRoadFactor
                          * penalty

            aer_structure = aer_structure + as * 2 * GR.MicroGridNS * DistRoadFactor
                          * penalty

            ManholeCost = ManholeCost + mh * 2 * GR.MicroGridNS * DistRoadFactor
                        * penalty
        end if

        i = i - 2       { run backbone down every other row }

    end loop (loop while (i >= rows_completed))


next n

grid_line_feet = (grid_line_feet / dist_lines_served) * total_lines


if UsePrimDist or (density < 100 PrimCutOffDensity) then
    prim_distribution_cost = zero
    prim_ugd_cable = zero
    prim_bur_cable = zero
    prim_aer_cable = zero
    prim_ugd_structure = zero
    prim_bur_structure = zero
    prim_aer_structure = zero
    prim_ManholeCost   = zero

    call calculate_prim_distribution_cost                    (primdist.pas)
    pass variables:
    GR
    number_of_SAIs
    SAIX
    SAIY
    density
    FillFactor
    lines
    flag
    *prim_distribution_cost
    *prim_line_feet
    *prim_drop_feet
    *prim_drop_cost
```



```
    *prim_nid_cost
    *prim_lines_served
    *prim_term_cost
    *prim_MaximumDistance
    *prim_ugd_cable
    *prim_bur_cable
    *prim_aer_cable
    *prim_ugd_structure
    *prim_bur_structure
    *prim_aer_structure
    *prim_ManholeCost



    if (ac_ugd_cop * prim_ugd_cable + ac_bur_cop * prim_bur_cable
      + ac_aer_cop * prim_aer_cable + ac_ugd_struc * prim_ugd_structure
      + ac_bur_struc * prim_bur_structure + ac_aer_struc * prim_aer_structure
      + ac_Manhole * prim_ManholeCost)

      <

      (ac_ugd_cop * ugd_cable + ac_bur_cop * bur_cable
      + ac_aer_cop * aer_cable + ac_ugd_struc * ugd_structure
      + ac_bur_struc * bur_structure + ac_aer_struc * aer_structure
      + ac_Manhole * ManholeCost)

    then
        grid_distribution_cost = prim_distribution_cost
        ugd_cable = prim_ugd_cable
        bur_cable = prim_bur_cable
        aer_cable = prim_aer_cable
        ugd_structure = prim_ugd_structure
        bur_structure = prim_bur_structure
        aer_structure = prim_aer_structure
        ManholeCost   = prim_ManholeCost
        grid_line_feet  = prim_line_feet
        grid_drop_feet  = prim_drop_feet
        grid_drop_cost  = prim_drop_cost
        grid_nid_cost   = prim_nid_cost
        grid_lines_served = prim_lines_served
        grid_terminal_cost = prim_term_cost
        MaximumDistance = prim_MaximumDistance
    end if

end if

{Now we need to handle the connection between the primary and secondary SAIs.}
{We minimize structure cost in connecting the SAIs using the algorithm suggested}
{by Prim, Bell System Technical Journal, 1957. }

call get_link_cost                                           (primsai.pas)
pass variables:
number_of_SAIS
SA
SAI_lines
saix
```

```
saly
density
*link_cost
*term_cost
*link_line_feet
*nc96
*nc24
*uc
*bc
*ac
*us
*bs
*as
*mh


grid_distribution_cost = grid_distribution_cost + link_cost + term_cost
ugd_cable = ugd_cable + uc
bur_cable = bur_cable + bc
aer_cable = aer_cable + ac
ugd_structure = ugd_structure + us
bur_structure = bur_structure + bs
aer_structure = aer_structure + as
ManholeCost   = ManholeCost   + mh
```

V - doesn't appear that distance for linking cables are carried forward

```
rocedure optimize_SAI_arrangement

passed variables:
SwitchX
SwitchY
GR
* SA
* grid_lines_served


local variables:
i
j
number_of_SAIs
X
Y
distribution_gauge
grid_distribution_cost
grid_line_feet
grid_drop_feet
density
drop_cost
drop_terminal_cost
nid_cost
mincost
link_cost
term_cost
```



```
nc96
nc24
MaximumDistance
uc
bc
ac
us
bs
as
mh
SAI_lines
link_line_feet


SA.SwitchX = GR.SwitchX
SA.SwitchY = GR.SwitchY
SA.ResLines = GR.gHouseholds * takerate * lines_per_house
SA.BusLines = GR.gBusinessLines
SA.lines = SA.ResLines + SA.BusLines
SA.DepthToBedrock = GR.DepthToBedrock
SA.Hardness = GR.Hardness
SA.SoilTexture = GR.SoilTexture
SA.WaterTb = GR.WaterTb
SA.MinSlope = GR.MinSlope
SA.MaxSlope = GR.MaxSlope
SA.quadrant = GR.quadrant

{ Calculate special access and switched DS1 lines }

SA.SpclAccessLines = SpclAccessRatio * GR.gBusinessLines
SA.SpclAccessDS1  = SA.SpclAccessLines * pct_les
SA.SpclAccessLines = SA.SpclAccessLines - SA.SpclAccessDS1
SA.SwitchedDS1 = (GR.gBusinessLines - SA.SpclAccessLines) * pct_del
GR.gBusinessLines = GR.gBusinessLines - round(SA.SwitchedDS1 + SA.SpclAccessDS1)
SA.Households = GR.gHouseholds


{ Need two copper lines for each DS1, which carries 24 channels }
```

*W - not sure what this accomplishes, units are lines^2/channel

```
SA.SpclAccessDS1 = SA.SpclAccessDS1 * 2/24
SA.SwitchedDS1 = SA.SwitchedDS1 * 2/24

SA.number_of_SAIs          = 1
SA.Grid_Distribution_Cost  = zero
SA.MaxDistance             = zero
SA.n2016                   = 0
SA.n672                    = 0
SA.n96                     = 0
SA.n24                     = 0
SA.nc96                    = 0
SA.nc24                    = 0
SA.anc96                   = 0
SA.anc24                   = 0
SA.fiber_terminal_cost     = zero
SA.t1_terminal_cost        = zero
SA.secondary_tterm_cost    = zero
SA.interface_cost          = zero
```

```
SA.drop_cost                - zero
SA.nid_cost                 - zero
SA.drop_terminal_cost       - zero
SA.grid_line_feet           - zero
SA.grid_drop_feet           - zero
SA.density                  - zero
SA.DepthToBedrock           - zero
SA.WaterTb                  - zero
SA.MinSlope                 - zero
SA.MaxSlope                 - zero
SA.DistToSwitch             - zero
SA.ugd_cable                - zero
SA.bur_cable                - zero
SA.aer_cable                - zero
SA.ugd_structure            - zero
SA.bur_structure            - zero
SA.aer_structure            - zero
SA.HanholeCost              - zero
SA.Density                  - GR.Density


for i = 1 to 4
    SA.SAI_lines[i] - zero
next

mincost - 1.0e+16

for number_of_SAIs - 1 to max_SAIs do
    grid_lines_served - zero

    SA.TypeOfSAI[1] - primary

    if number_of_SAIs > 1 then

        for i - 2 to number_of_SAIs
            SA.TypeOfSAI[i] - secondary
        next

        case number_of_SAIs
           case 1
              x[1] - GR.cx1[1]
              y[1] - GR.cy1[1]

           case 2
               for i - 1 to 2
                   x[i] - GR.cx2[i]
                   y[i] - GR.cy2[i]
               next

           case 3
               for i - 1 to 3
                   x[i] - GR.cx3[i]
                   y[i] - GR.cy3[i]
               next

          case 4
               for i - 1 to 4
```



```
                   x[i] - GR.cx4[i]
                   y[i] - GR.cy4[i]
               next

        end case

        for i - number_of_SAIs + 1 to 4
            x[i] - GR.cx4[i]
            y[i] - GR.cy4[i]
        next

        grid_distribution_cost - zero

        call calculate_grid_distribution_cost                    (distrib.pas)
        pass variables:
        GR
        SA
        number_of_SAIs
        X
        Y
        *SAI_lines
        *grid_distribution_cost
        *grid_line_feet
        *link_line_feet
        *grid_drop_feet
        *density
        *drop_cost
        *drop_terminal_cost
        *nid_cost
        *grid_lines_served
        *link_cost
        *term_cost
        *nc96
        *nc24
        *MaximumDistance
        *uc
        *bc
        *ac
        *us
        *bs
        *as
        *mh


        if (ac_ugd_cop * uc + ac_bur_cop * bc + ac_aer_cop * ac
          + ac_ugd_struc * us + ac_bur_struc * bs + ac_aer_struc * as
          + ac_manhole * mh + ac_tl_term * term_cost) < mincost then

            mincost - ac_ugd_cop * uc + ac_bur_cop * bc + ac_aer_cop * ac
                    + ac_ugd_struc * us + ac_bur_struc * bs + ac_aer_struc * as
                    + ac_manhole * mh + ac_tl_term * term_cost

            SA.number_of_SAIs          - number_of_SAIs
            SA.X                       - X
            SA.Y                       - Y
            SA.grid_distribution_cost  - grid_distribution_cost - term_cost
            SA.secondary_tterm_cost    - term_cost
```

```
            SA.snc96                    = nc96
            SA.snc24                    = nc24
            SA.grid_line_feet           = grid_line_feet
            SA.link_line_feet           = link_line_feet
            SA.grid_drop_feet           = grid_drop_feet
            SA.density                  = density
            SA.drop_cost                = drop_cost
            SA.drop_terminal_cost       = drop_terminal_cost
            SA.mid_cost                 = mid_cost
            SA.MaxDistance              = MaximumDistance
            SA.ugd_cable                = uc
            SA.bur_cable                = bc
            SA.aer_cable                = ac
            SA.ugd_structure            = us
            SA.bur_structure            = bs
            SA.aer_structure            = as
            SA.ManholeCost              = mh
            SA.lines_served             = grid_lines_served

            for i = 1 to 4
                SA.SAI_lines[i] = SAI_lines[i]
            next
        end if
    end if

next number_of_SAIs
```


feeder.pas

```
feeder.pas

the only procedure used outside of this module is optimize_feeder_arrangement


function L1_distance  ,
    variables passed in:
    x1
    x2
    y1
    y2

    L1_distance = abs(x1 - x2) + abs(y1 - y2)



function L2_distance
    variables passed in:
    x1
    x2
    y1
    y2


    local constants:
    KFPerStatMi = 5.28
    StatMiPerMin = 1.1515
    MinPerDegree = 60

    local variables:
    degY1
    degY2
    degX1
    degX2
    cosa
    alpha

    degY1 = y1 / (KFPerStatMi * StatMiPerMin * MinPerDegree) + CR.OriginY
    degY2 = y2 / (KFPerStatMi * StatMiPerMin * MinPerDegree) + CR.OriginY
    degX1 = x1 / (KFPerStatMi * StatMiPerMin * MinPerDegree
          * cos(CR.reference_latitude * pi / 180)) + CR.OriginX

    degX2 = x2 / (KFPerStatMi * StatMiPerMin * MinPerDegree
          * cos(CR.reference_latitude * pi / 180)) + CR.OriginX

    { Here is the formula to calculate the great circle arc, taken from }
    { Love, Morris, and Wesolowsky: }


    cosa = cos(degY1 * pi / 180) * cos(degY2 * pi/180)

    cosa = cosa * cos((degX1 - degX2) * pi / 180)

    cosa = cosa + sin(degY1 * pi / 180) *sin(degY2 * pi / 180)
```

```
    if (abs(cosa) > 1.0e-6) then
        alpha = ArcTan(sqrt(1 - cosa * cosa) / cosa) * 180 / pi
    else
        alpha = 90.0
    end if

    L2_distance = abs(alpha * MinPerDegree * StatMiPerMin * KFPerStatMi)


rocedure optimize_feeder_arrangement
    variables passed in:
    SwitchX
    SwitchY
    density
    num_SAs
    *SA_array
    *feeder_cost
    *feed_splice_cost
    *ugd_cable
    *bur_cable
    *aer_cable
    *ugd_fiber
    *bur_fiber
    *aer_fiber
    *ugd_structure
    *bur_structure
    *aer_structure
    *ManholeCost


    local variables:
    i
    j
    k
    n
    dmtx
    x
    y
    _from
    _to
    cut_ord
    DistToNode
    DistToSwitch
    feeder_distance
    tcost
    cable_cost
    structure_cost
    FillFactor
    primfile
    uc
    bc
    ac
    uf
    bf
    af
```



```
    us
    bs
    as
    mh
    ac_structure
    pct_ugd
    pct_bur
    pct_aer
    xlon
    ylat


    this is a procedure within the procedure

    procedure calculate_feeder_structure_costs
          passed variables:
          SwitchX
          SwitchY
          density
          *structure_cost
          *FillFactor
          *ugd_structure
          *bur_structure
          *aer_structure
          *ManholeCost
          *pct_ugd
          *pct_bur
          *pct_aer


          local variables:
          i
          totlines
          fib_lines
          cop_lines
          technology
          n2016
          n672
          n96
          n24
          us
          bs
          as
          mh
          pu
          pb
          pa


          { This procedure calculates an average structure cost for the feeder based
            on average line density for the feeder network (given).}


          totlines = zero
```

```
for i = 1 to num_SAs
    totlines = totlines + SA_array[i].lines
next i

if totlines > 0 then
    density = density / totlines
else
    density = zero
end if

Fillfactor = Fill_Factor_fn(density,1)                    (global.pas)

for i = 1 to NumDensZones
    if (density >= CopFeedPlantMix[i].density) then
        pct_ugd = CopFeedPlantMix[i].UgdPct
        pct_bur = CopFeedPlantMix[i].BurPct
        pct_aer = CopFeedPlantMix[i].AerPct
    end if
next i


if (pct_ugd + pct_bur + pct_aer) < one then

    {provisionally, assign half of "free" percentage to aerial, half to buried }

    pct_aer = pct_aer + half * (one - pct_ugd - pct_bur - pct_aer)
    pct_bur = pct_bur + half * (one - pct_ugd - pct_bur - pct_aer)

end if
```

V - this is inconsistent with the way distribution does this same thing, dist'n assigns extra to portion with largest cost

```
{ Now approximate feeder technology based on distance to switch }

for i = 1 to num_SAs
    //not sure what DistanceType is
    if DistanceType = 1 then
        feeder_distance = call L1_distance * FeederRoadFactor      (feeder.pas)
                          pass variables:
                          X1 = SA_array[i].x[1]
                          X2 = SwitchX
                          Y1 = SA_array[i].y[1]
                          Y2 = SwitchY
    else
        feeder_distance = call L2_distance * FeederRoadFactor      (feeder.pas)
                          pass variables:
                          X1 = SA_array[i].x[1]
                          X2 = SwitchX
                          Y1 = SA_array[i].y[1]
                          Y2 = SwitchY
    end if


    call calculate_feeder_technology                               (tech.pas)
    pass variables:
    feeder_distance
    i
```



```
    density
    Fillfactor
    *technology
    *n2016
    *n672
    *n96
    *n24
    pct_ugd
    pct_bur
    pct_aer

next i


{ Now calculate line-weighted average structure cost and pct bur, aer, ugd. }

structure_cost = zero
ugd_structure  = zero
bur_structure  = zero
aer_structure  = zero
ManholeCost    = zero
pct_ugd        = zero
pct_bur        = zero
pct_aer        = zero

for i = 1 to num_SAs
    if SA_array[i].feeder_technology = fiber then
        fib_lines = (SA_array[i].n2016 + SA_array[i].n672 + SA_array[i].n96
                   + SA_array[i].n24) * 4

        structure_cost = structure_cost + SA_array[i].lines
                       * call structure_cost_fn                    (structur.pas)
                         pass variables:
                         0
                         fib_lines
                         density
                         SA_array[i].hardness
                         SA_array[i].DepthToBedrock
                         SA_array[i].SoilTexture
                         SA_array[i].MinSlope
                         SA_array[i].MaxSlope
                         SA_array[i].WaterTb
                         1
                         0
                         1
                         *us
                         *bs
                         *as
                         *mh
                         *pu
                         *pb
                         *pa


        ugd_structure = ugd_structure + SA_array[i].lines * us
        bur_structure = bur_structure + SA_array[i].lines * bs
```

```
    aer_structure = aer_structure + SA_array[i].lines * as
    ManholeCost = ManholeCost + SA_array[i].lines * mh
    pct_ugd = pct_ugd + SA_array[i].lines * pu
    pct_bur = pct_bur + SA_array[i].lines * pb
    pct_aer = pct_aer + SA_array[i].lines * pa

else if SA_array[i].feeder_technology = t_1 then
    cop_lines = SA_array[i].lines / 12
    structure_cost = structure_cost + SA_array[i].lines
                   * call structure_cost_fn                    (structur.pas)
                     pass variables:
                     cop_lines
                     0
                     density
                     SA_array[i].hardness
                     SA_array[i].DepthToBedrock
                     SA_array[i].SoilTexture
                     SA_array[i].MinSlope
                     SA_array[i].MaxSlope
                     SA_array[i].WaterTb
                     1
                     1
                     0
                     *us
                     *bs
                     *as
                     *mh
                     *pu
                     *pb
                     *pa

    ugd_structure = ugd_structure + SA_array[i].lines * us
    bur_structure = bur_structure + SA_array[i].lines * bs
    aer_structure = aer_structure + SA_array[i].lines * as
    ManholeCost = ManholeCost + SA_array[i].lines * mh
    pct_ugd = pct_ugd + SA_array[i].lines * pu
    pct_bur = pct_bur + SA_array[i].lines * pb
    pct_aer = pct_aer + SA_array[i].lines * pa

else
    structure_cost = structure_cost + SA_array[i].lines
                   * call structure_cost_fn                    (structur.pas)
                     pass variables:
                     SA_array[i].lines
                     0
                     density
                     SA_array[i].hardness
                     SA_array[i].DepthToBedrock
                     SA_array[i].SoilTexture
                     SA_array[i].MinSlope
                     SA_array[i].MaxSlope
                     SA_array[i].WaterTb
                     1
                     1
                     0
                     *us
                     *bs
```



```
                     *as
                     *mh
                     *pu
                     *pb
                     *pa

    ugd_structure = ugd_structure + SA_array[i].lines * us
    bur_structure = bur_structure + SA_array[i].lines * bs
    aer_structure = aer_structure + SA_array[i].lines * as

    ManholeCost = ManholeCost + SA_array[i].lines * mh
    pct_ugd = pct_ugd + SA_array[i].lines * pu
    pct_bur = pct_bur + SA_array[i].lines * pb
    pct_aer = pct_aer + SA_array[i].lines * pa
  end if
next i

structure_cost = structure_cost / totlines
ugd_structure = ugd_structure / totlines
bur_structure = bur_structure / totlines
aer_structure = aer_structure / totlines
ManholeCost = ManholeCost / totlines
pct_ugd = pct_ugd / totlines
pct_bur = pct_bur / totlines
pct_aer = pct_aer / totlines


procedure reverse_convert
    passed variables:
    xkf
    ykf
    reflon
    reflat
    *xlon
    *ylat

    local constants:
    EarthRadius_meters = 6367723
    KFPerMeter = 0.00328083989501312

    local variables:
    NSCirc
    EWCirc

    NSCirc = 2 * pi * EarthRadius_meters * KFPerMeter

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

    ylat = ykf * 360 / NSCirc + reflat
    xlon = xkf * 360 / EWCirc + reflon


start of procedure calculate_feeder_structure_costs
```

```
{ First, calculate structure costs to be used in tree calculation. }

call calculate_feeder_structure_costs                    (feeder.pas)
pass variables:
SwitchX
SwitchY
density
*structure_cost
*fillFactor
*us
*bs
*as
*mh
*pct_ugd
*pct_bur
*pct_aer


ac_structure = ac_ugd_struc * us + ac_bur_struc * bs + ac_aer_struc * as
             + ac_manhole * mh


{ Now we need to set up the distance matrix to be used by PrimTree. }

n = num_SAs + 1
x[1] = SwitchX
y[1] = SwitchY

for i = 2 to n
    x[i] = SA_array[i-1].x[1]
    y[i] = SA_array[i-1].y[1]
next

for i = n+1 to n + num_SAs
    x[i] = SA_array[i-1-num_SAs].x[1]
    y[i] = SwitchY
next

for i = n + num_SAs + 1 to n + num_SAs + num_SAs
    x[i] = SwitchX
    y[i] = SA_array[i-1-2*num_SAs].y[1]
next

n = n + 2 * num_SAs

for i = 1 to n

    for j = 1 to i
        if DistanceType = 1 then
            dmtx[i][j] = call L1_distance * FeederRoadFactor      (feeder.pas)
                         pass variables:
                         X1 = x[i]
                         X2 = x[j]
                         Y1 = y[i]
                         Y2 = y[j]

        else
            dmtx[i][j] = call L2_distance * FeederRoadfactor      (feeder.pas)
```



```
                         pass variables:
                         X1 = x[i]
                         X2 = x[j]
                         Y1 = y[i]
                         Y2 = y[j]

            dmtx[j][i] = dmtx[i][j]
        end if
    next j
next i


call prim_tree                                           (primfeed.pas)
pass variables:
n
dmtx
density
fillFactor
ac_structure
*_from
*_to
*DistToNode
*DistToSwitch
pct_ugd
pct_bur
pct_aer

open the file primfile with filename = 'prim.asc'

write to primfile: '_from      _to DtoNod  DToSw      x       y'

for i = 1 to n
    call reverse_convert                                  (feeder.pas)
    passing variables:
    x[i]
    y[i]
    CR.OriginX
    CR.reference_latitude
    xlon
    ylat

    write to primfile: _from[i], ' ', _to[i], ' ', DistToNode[i], ' ',
                       DistToSwitch[i], ' ', xlon, ' ', ylat

    call reverse_convert                                  (feeder.pas)
    passing variables:
    x[_to[i]]
    y[_to[i]]
    CR.OriginX
    CR.reference_latitude
    xlon
    ylat

next


{ Determine cuts by pruning branches }
```

```
call prune                                          (primfeed.pas)
pass variables:
n = n
_to = _to
*cut_ord = cut_ord


{Initialize disaggregated structure cost to per-kf values.  Cumulate function will}
{return total disaggregated structure costs. }

ugd_structure = us
bur_structure = bs
aer_structure = as
ManholeCost = mh

{ Accumulate lines at each node, and sum total feeder cost }


call cumulate_lines                                 (primfeed.pas)
pass variables:
n
_to
DistToNode
DistToSwitch
cut_ord
structure_cost
density
FillFactor
*feeder_cost
*ugd_cable
*bur_cable
*aer_cable
*ugd_fiber
*bur_fiber
*aer_fiber
*ugd_structure
*bur_structure
*aer_structure
*ManholeCost
pct_ugd
pct_bur
pct_aer


for i = 1 to num_SAs
    SA_array[i].DistToSwitch = DistToSwitch[i+1]
next
```



```
structur.pas


the only procedure used outside of this module is structure_cost_fn


function structure_cost_fn

    passed variables:
    copper_lines
    fiber_lines
    density
    hardness
    depth_to_bedrock
    soil_texture
    MinSlope
    MaxSlope
    WaterTb
    feeder_indicator { tells us this is for feeder }
    copper_indicator  { tells us there is copper }
    fiber_indicator   { tells us there is fiber }
    *ugd_structure
    *bur_structure
    *aer_structure
    *manhole_cost
    *pct_ugd
    *pct_bur
    *pct_aer



    local variables:
    i
    critical_depth
    soil_texture_indicator
    fiber_cables
    copper_cables
    NumberOfDucts
    ManholeSpacing
    ugd_share
    bur_share
    aer_share
    free_pct


    pct_ugd                 = zero
    pct_bur                 = zero
    pct_aer                 = zero
    critical_depth          = zero
    soil_texture_indicator  = 0
    fiber_cables            = 0
    copper_cables           = 0
    NumberOfDucts           = 0
    ManholeSpacing          = zero
    ugd_share               = zero
    bur_share               = zero
    aer_share               = zero
```